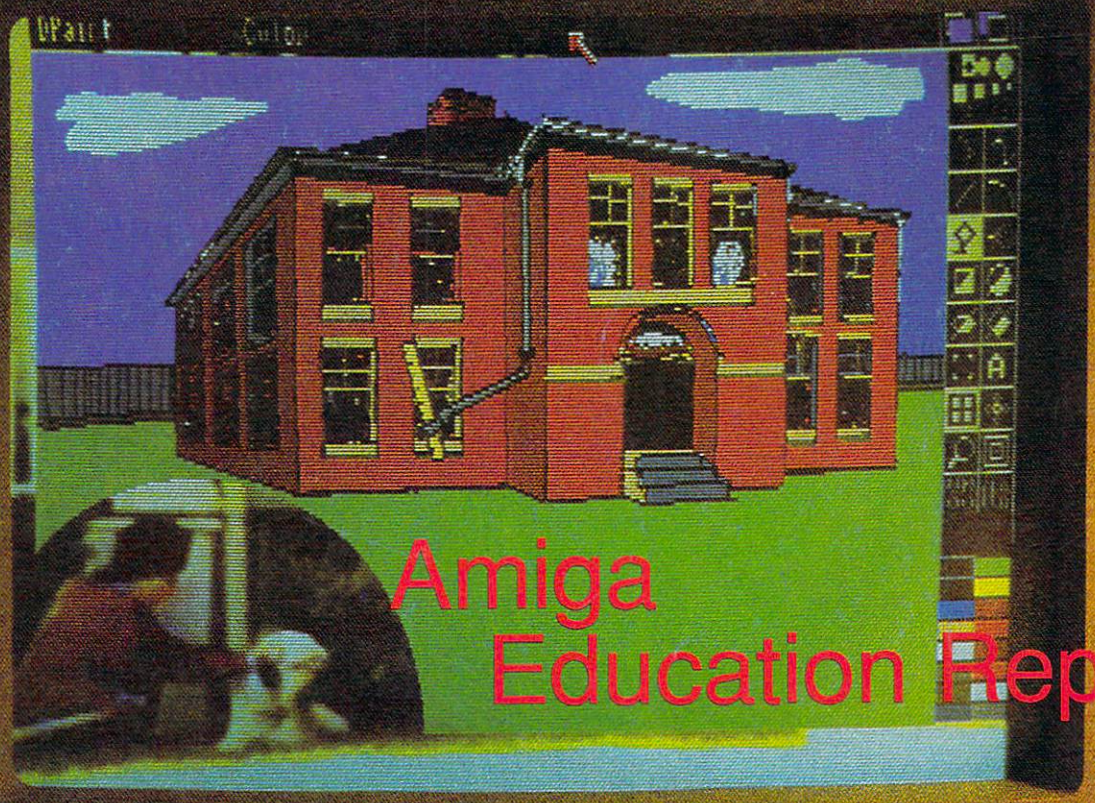
Using Your Printer From Basic by Sheldon Leemon

Amazing Computing™

Volume 1 Number 8
U.S.A. $3.50
Canada $4.50

Commodore Amiga™ Information and Programs



Amiga Education Report

Reviews:
- EMACS
- Marble Madness
- Scribble!
- True Basic
- Lattice MAKE Utility

Screen Saver
Using Fonts in Basic

Superbase™ PERSONAL
RELATIONAL DATABASE SYSTEM

SUPERBASE IS NOW AVAILABLE FOR THE AMIGA

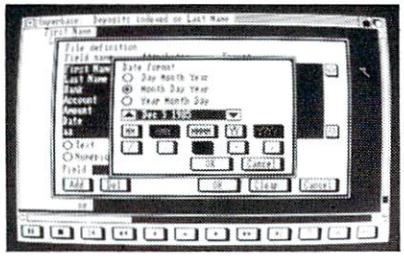
The enormously popular and proven database system, Superbase is now available for the Amiga computer. We completely rewrote Superbase to take full advantage of all the power available on the amazing Amiga. This is not a conversion, but an entirely new Amiga program!

TOTAL SOLUTIONS

Superbase provides the total information management solution. It is a true productivity program for the Amiga computer. You can finally use a serious database with a serious machine. It's easy to keep track of inventories for your business whether you're working with parts inventory or real estate listings. Superbase is perfect for church membership rolls, patient files, personnel schedules or any place you need to manage and control large amounts of important information.

Access the power of the first **true** relational database management system with Superbase Personal Relational Database System. It will turn your Amiga into a truly productive tool, with virtually limitless capacities. Imagine being able to have an unlimited number of files open at any time. You can even have each file indexed with up to 999 key fields to search and sort.

EASY TO SET UP



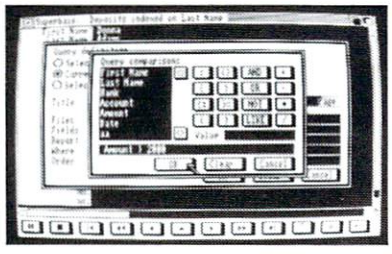
Superbase utilizes the latest ideas in easy-to-use mouse and windowing technology. There are pull-down help menus to ease you through problems that may occur during database creation. Superbase is completely menu driven and takes advantage of the point-to-click features possible with the Amiga mouse.

Create a database in minutes using the easy to understand menu selections and control panels. Type in field names, add details like length or data style and you are quickly ready to input your data. Unlike other databases, you can alter your formats at anytime, **without disturbing** the data already in existing files. Using Superbase's Enhanced BASIC, your database can be totally customized to virtually any application.

IT'S EASY TO MANAGE YOUR DATA

Display your data in the format you choose. Either page by page or just as it appears in the record format. You choose how to view the data you need. There is practically no limit to the number of fields in a record, you have complete control over what is displayed on screen or printed in custom reports.

Decide on the fields and on the sequence, then use the VCR style controls to view your data -- Get the first record, then fast forward, pause, continue or stop -- it's as easy as playing a video tape!



WORKING POWER

Superbase makes it easy to define reports or generate relational queries across multiple files, with multiple sort levels if you need them. Import data from other databases or applications. Export data to your favorite word processor, or join several files to form a new database,

The advanced B+ tree file structure and disk buffering means high performance -- Superbase reads a typical name and address record in an incredible three hundredths of a second!

THE VIDEO DATABASE

Superbase includes an amazing array of data types in your record format, including the names of pictures or digitized images stored on disk. Read the words, then look at digitized pictures you have already stored on disk. Your data records can "point" to images to recall them for viewing!

You can even link multiple images to a single database record to run automatic slide shows. It's all easily done using the VCR style commands that you control. Revise, update or review your illustrated database in any desired arrangement. You have total control! Superbase is the total software solution for people who must manage information.

THE BEST HAS ARRIVED!

Finally, a program that utilizes ALL the power and functions of the Amiga computer. Superbase brings to the Amiga the business solutions you have been waiting for.

The power of Superbase is also available for the Commodore 64/128 and the Apple IIe/IIc.

WHEN QUALITY COUNTS!

PROGRESSIVE PERIPHERALS & SOFTWARE

464 KALAMATH STREET
DENVER, COLORADO 80204
303-825-4144
TELEX: 888837

Amazing Dealers

The following are Amazing Dealers, dedicated to supporting the **Commodore-Amiga™**. They carry **Amazing Computing™**, your resource for information on the Amiga™.

If you are not an Amazing Dealer, but would like to become one, contact:

PiM Publications, Inc.
P.O. Box 869
Fall River, MA. 02722
1-617-678-4200

Alabama
Abex Data Systems — Huntsville
Command Computers — Birmingham
Madison Books & Computers — Madison
Mail Tel Communications — Mobile
Mel's Photo & Computer Shop — Montgomery
Universal Computer System — Mobile

Arkansas
SIS Inc. — Mt. Home
The Micro Shop — Little Rock

Arizona
Computer Library, Inc. — Tempe
Computer West — Phoenix
Copperstate Business Systems — Phoenix
Copperstate Computers — Tuscon
North American Digital — Tuscon
Tronixtools — Tuscon

California
Bits n Pieces Comp Prod. — Anahiem
Brown Knows Computing — Redlands
C.F.T. — Claremont
Century Computer Systems — La Habra
Computer Attic — Palo Alto
Computer Junction — Senata Ana
Computer Literacy Bookshop — Sunnyvale
Computer Nook, Inc. — San Bernadino
Computer Technology — Pomona
ComputerLand Fresemont — Freemont
ComputerLand of Stockton — Stockton
Dataphile — Watsonville
Gary Bell — Santee
Harding Way News — Stockton
Home Computing Center — Mill Valley
Home Computing Center — San Bruno
Home Computing Center — San Leanardo
K J Computers — Granada Hills
K J Computers — Granada Hills
Levity — North Hollywood
Micro Flight — San Jacinto
Op Amp Tech Books — Los Angeles
Ridgecrest Computer Ctr. Inc. — Ridgecrest
S.O.S. Computers — Los Angeles
Software First — Santa Rosa
Software Plus — Citrus Heights
The Computer Room — Scotts Valley
The Floppy Disk Inc. — Downey
Winner's Circle — Berkley
HT Electronics — Sunnyvale

Colorado
Citadel Computers — Colorado Springs
Computer Discount — Denver
Computer Room — Aurora
Micro World — Lakewood
Sun Country Computers — Colorado Springs
Whole Life Distributors — Englewood
Whole Life Distributors — Englewood

Connecticut
Mnemonics Inc. — Norwalk
Personal Computer Center — Norwich
Software City — Orange
Software Kingdom — East Windsor
Spectrum Computer Services — New Haven

Delaware
Castle Video — Newark

Florida
A A Computers — Jacksonville
Book Mania — Orlando
Commodore Computer Shop — Ocala
Computer Bar — Pennsacola
Computer Base Enterprises Inc. — Tampa
Computer Image — Miami
Computer Terminal Inc. — Sebring
Computers 4 Rent — W. Palm Beach
Computers Plus — South Daytona
Cotronics Inc. — Stewart
Education Computers Etc. — Tallahasee
Family Computers — Sarasota
Florida Book Store — Gainsville
Gulf Coast Computer Exchange — Panama City
Megaport Computer Center — South Venice
Micro Computer Center — Largo
Micro's Etc. — Altamonte Springs
MSI Business System — Brooksville
New Age Electronics — St. Petersburg
Rainbow Computr Center — Orlando
Random Access Computers Inc. — Ft. Walton Beach
Softerals Computer Center Inc. — Holly Hill
The Open Door — Cocoa
The PC Collection — Coral Gables

Georgia
Future System — Augusta
Michael Donnellan — Decatur
Software South Inc. — Savannah
The 64 Store — Atlanta

Hawaii
Computer House Inc. — Honolulu

Idaho
ABI Computer & Video — Nampa

Illinois
BA Business Computers — Batavia
Computer Resolution — Champaign
Computerland O'Hare — Park Ridge
ComputerLand of Niles — Niles
Elmi Microcomputers Inc. — Naperville
Interstate Computer Service — LeMont
The Memory Expension — Normal
Unique Computer — Sterling
Vincent Software Systems — Fairview Hts.

Indiana
Burkat Computer Ctr. — South Bend
Bytrax Computer — Ft. Wayne
Computer Corner — Fort Wayne
Computer Corner — Marion
Computer People Inc. — Michigan City
Greg Chaney — Greenfield
Micro Computers Inc. — Indianapolis
Spectra Computer Sys. Inc. — Evansville
Vons Computers — W. Lafayette

Iowa
Century Systems — Desmoine
Micro Computer Applications LTD — Marshalltown

Kansas
Data Bank Corp. — Wichita
J & L Electronics — Liberal
MidKansas Computers — Newton
Midwest Computers — Manhattan
Thorotrad Computers — Topeka

Kentucky
MicroAge Computer Store — Lexington
Reality World Computers — Lexington

Louisiana
Computer Time — Shreveport
Modern Business Machines — Metairie
Software Center International — Metairie
The Computer Clinic — Lafayette

Maine
Computer Barn — Damariscotta
Valley Computers — Auburn

Maryland
Computer Crafters — Wheaton
ComputerWorld — Ocean City
Compuvision Vision Center — Columbia
Micro Computer Center — Baltimore
Software Advantage — Rockville
Software N Things — College Park
The Logical Choice Inc. — Baltimore
Waldorf Computer — Waldorf

Massachusetts
Club Computer — Cambridge
E.U. Wurlitzer Co. — Boston
General Computer Store — Framingham
HCS Computer Center — Marshfield
LCA Video & Computer Ctr. — Norwood
Memory Location — Wellsley
Metro Computer — Brookline
OmniTek Computers — Tewksbury
Pioneer Valley Data Equipment — Amherst
Tech Computer Store — Cambridge
The Bit Bucket — West Newton
Tycom Inc. — Pittsfield

Michigan
Basic Computer Center — Grand Rapids
Bits-Bytes-Nibbles — Petoskey
Canton Computer — Canton
CompuSoft — Kalamazoo
Computer Supply Co. — Traverse City
Direct Access — Lansing
Edmunton Computer Center — Pontiac
Galaxy Computers — Allen Park
Global Computer Center — Derborn
Lavassee Computer Inc. — Adrian
Michigan Software — Novi
Midland Computer Shop — Midland
Pro-Video — Okemos
Professional Computer Systems — St. Joseph
Programs Unlimited — Grand Rapids
Retail Computer Center Inc. — Farmington Hills
Roseville Computer Store — Warren
Skip Disk — Madison
Heights
Software Plus — Grand Rapids
Specialists Computer — Livonia
State Street Computers — Ann Arbor
The Software House — Kalamazoo
Ye Olde Computer Shoppe — Ypsilanti

Minnesota
Computer Outfitters — Roseville
Computer Place — Minneapolis
Computer Specialties Plus — Monticello
Computers, etc. — Eagan
JMH Software — Maple Grove
MCD of Hibbing Inc. — Hibbing
OnLine Computers — Minnetonka
Specialists IN — Hopkins

Missouri
Associated Computer Services — Springfield
Brands Mart Computers — Kansas City
Data Byte Computer Co. — St. Louis
Instant Replay Ltd. — Chesterfield
Systems Plus — Independence
Systems Plus of St. Louis — St. Louis

Mississippi
Enterprises Unlimited — Jackson
National Computer Center — Ocean Springs

Montana
Appelgren Computer Systems — Great Falls

Nebraska
BULLS-EYE — Lincoln
My Com Inc. — Plattsmouth
Double E Electronics — Omaha

Nevada
Century 23 Inc. — Las Vegas
Computer World — Las Vegas
ComputerLand of Carson — Carson City

New Hampshire
Computer Mart of New Hampshire — Nashua
Diversified Computers — Keene

New Jersey
Family Computer Centres — Fairfield
Family Computer Centres — South Orange
Family Computing Centres — Ridgewood
Morris County Stationers — Chester

New Mexico
Academy Computers — Albuquerque
New Horizon Computer System — Alamogordo
Page One Newstand — Albuquerque
Technical Concepts — Las Crues

New York
Amcus- Hudson Valley — Wappingers Falls
Amuse — New York
Byte Shop — Merrick
Castle Computer — Latham
CIA Software Center — Flushing
Compurite Center — White Plains
Computer Outlet — Jamestown
Computers Etc. — Syracuse
Leigh's Computers — New York
Ray Supply Inc. — Glen Falls
Software Centre International — Forest Hills
Software & Such — Scotia
Software City — Forest Hills
Software Supermarket — Kenmore
Star Tech Systems — Massena
Tedrow Business Products — Rochester
The Working Computer — Stoney Brook
Video Computer Bin — Brooklyn
Video Computer Center — Rome
World Computers Inc. — Hicksville

North Carolina
Digitz — Raleigh
TDS Computers — Durham
TDS Computers — Carrboro
Triad Computers — Greensboro

North Dakota
Computer Associates — Fargo
The Computer Store — Minot

Ohio
Computer Network — Newark
Computer Site — Cleveland
Computers Plus of Ohio — Upper Sandusky
Earthrise Micro Systems — Columbus
Lakes Consumer Electronics — Akron
Microwave Magic — Fairfield
North Coast Programming — Willoughby
Program Store — Columbus
Quality Computer Appliances — Toledo
Quality Computer Appliances — Toledo
Saxtan Computer Center — Sandusky
Software Centre International — N.Olmsted
Software and More — Cincinnati
Software Connection — Mayfield Hts.

Oklahoma
Colonial Video & Computer — Bartlesville
P C Tech — Tulsa
Second Hand Software — Oklahoma City
Video-Comp Inc. — Lawton

Oregon
Clackamas Computers — Clackamas
Clackamas Computers — Beaverton
Computron Business Systems — Portland
IB Computers — Portland
Software Center — Beaverton
Software Express — Eugene
University of Oregon Bookstore — Eugene

Pennsylvania
Alpha Omega EBE — Warren
Basic Computer Systems — Hermitage
Data Sofique Computerware — Pittsburg
Downingtown Computer Center — Downingtown
East Coast Software — Hershey
Pittsburgh Computer Store — Pittsburgh
The Electronic Boutique — Norristown
Triangle Computers — Indiana

Rhode Island
Micro Limits — Smithfield
International Computer Services — Johnston
Micro Limits — Warwick

South Carolina
Avcom International Inc. — West Columbia
Horizon — Greenville
Micro Computer Depot — Sumter

South Dakota
Computer Business Center — Rapid City

Tennessee
ComputerLab of Memphis — Memphis
Software First — Nashville

Texas
Byte by Byte — Austin
C.E.M. Corporation — Brazoria
Colonial Video — Houston
Computer Age — Dallas
Computer Age — Houston
Computer Concepts — Waco
Computer Magic — Austin
Computer Revelations — Houston
Computer Time Inc. — Marshall
Metropolitan Computer — Richardson
Micro Search — Huston
Regency Educational Systems — Dallas
Softex — Watauga
Software Centre — Dallas
Software Library — Wichita Falls
Software Terminal — Fort Worth
The Computer Experience — San Antonio
The Computer Stop — Abalean

Utah
Computers Plus — Murray

Virginia
Colony Ltd. — Hampton
Diskovery — Falls Church
Family Computer Center — Fairfax
Micro Enhancement — Virginia Beach
Software Central — Norfolk
Virginia Micro Systems — Woodbridge

Washington
A.P.P.L.E. CO-OP — Renton
Apple Valley Computers — E. Wentachee
Com-Soft Computers — Everett
Computers + Inc. — Tacoma
Micro Age Computer Store — Bellview
Nibble Site — Tocoma
Omni International — Seattle
Programs Plus — Seattle
Take-A-Byte — Kennewick

Wisconsin
Colortron Computers — Racine
Computer Software Center — Milwaukee
Mom's Computers — Stevensport
Software Shoppe — Fond Du Lac
TMW Software Inc. — Wausa

Canada
Compute or Play — Alberta
Computerworks — Alberta
Computer Shop — Alberta
Data Pro — Alberta
Computer Shop of Calgary — Alberta
The Software House — Alberta
Hindsin Computing — Alberta
Red Deer Micro Systems — Alberta
Memory Land — Alberta
Computer King — British Columbia
Conti — British Columbia
Nutec — British Columbia
Sprite — British Columbia
Strider — British Columbia
Super Software — British Columbia
Canadian Computer Era — Manitoba
Micromart — Manitoba
ComputerWorld — Maritimes
J.W. Allen — Maritimes
Kobetek — Maritimes
Minicomp — Maritimes
Arkon — Ontario
Chatam Computer — Ontario
Computerland — Ontario
Comsoperia — Ontario
Electronics 2001 — Ontario
Hastings Data — Ontario
The Computer & You — Ontario
Thornhil — Ontario
Software Co. — Quebec
Daltronic Solare — Quebec
Microdata — Quebec
Info Mersier — Quebec
Laval Micro — Quebec
Vision Tronique — Quebec
Distribution Maipro — Quebec

Distributed in Canada by:
Avita Software Distributors — Concord, Ont.
Compulit Distributors — British Columbia
Phase 4 Dist. Inc. — Calgary, Alberta

Japan
Pineapple 6502 — Tokyo

Amazing Computing™

Publisher: Joyce Hicks
Circulation Manager: Doris Gamble
Assistant to the Publisher:
Robert James Hicks
Corporate Advisor: Robert Gamble

Managing Editor: Don Hicks
Hardware Editor: Ernest P. Viveiros Sr.
Amicus & Technical Editor: John Foust
Music Editor: Richard Rae
Art Director: Keith Conforti
Assistant Editor: Ernest P. Viveiros Jr.
Assistant Advertising Manager:
John David Fastino

Amazing Authors:
Ervin Bobo
Bryan Catley
John Foust
Don Hicks
Kelly Kauffman
Perry Kivolowitz
George Musser Jr.
Steven Pietrowicz
Rick Wirch
&
The Amigo

Special Thanks to:
Robert H. Bergwall
RESCO, Inc.
E.P.V. Consulting
New England Technical Services
Interactive Tutorials Inc.

Advertising Sales
&
Editorial

1-617-678-4200

Amazing Computing™ (ISSN 0886-9480) is published by PiM Publications, Inc., P.O. Box 869, Fall River, Ma. 02722. Subscriptions: in the U.S. 12 Issues for $24.00; Canada and Mexico, $30.00; Overseas, $35.00. Printed in the U.S.A.

First Class or Air Mail rates available upon request.

PiM Publications maintains the right to refuse any advertising.

Amazing Computing™

Amazing Contents

Volume 1, Number 8 1986

From The Editor:

Education
This month, Amazing Computing™ is dedicated to Education and the Amiga. Although, as in all other aspects of this machine, the software is still being developed for home and school, we found some nifty items just waiting to be seen.

There is a small section devoted to MicroEd and Torry Esbensen. A man who has made a large contribution to the world of education on the Amiga.

Geoff Gamble reports on the work being performed at Washington State University by a diverse yet active group of users. In the course of preparing this report, Geoff established a user group at the University.

There are several reviews of educational products and, as a teacher, Robert Frizelle asks fellow teachers to embrace this new technology.

Why Education?
This is our Educational issue, but, in truth, all of our Amazing Computing™ issues hope to be educational. We intend to impart a great deal more than just information on the next best game or spreadsheet.

We do like to research games such as Marble Madness, however, we will also review products such as True Basic.

Reviews are important and we will continue to provide information on all of the fast new items coming for your Amiga. However, reviews do not tell you the how or why of your machine. For that, we have turned to the authors actually working with the machines every day. Thus, we have included Screen SaVer, Using your printer with the Amiga, Using fonts from Amiga Basic and .bmap reader.

Our authors are real people with real jobs who also have a hankering to stay up until 3:00 AM searching for an ellusive bug in an intricately designed program. They spend hours improving the Amiga so it will perform a task for them a little quicker than before.

These people are dedicated. They think nothing of spending hours hunched over some obscure documentation, attempting to find an answer to their questions.

In short, they are students.

We all are.

We have been studying the Amiga with an intensity that, if applied in our "younger" years, would have thrilled our teachers and our parents.

Yet, we are applying the techniques education gave us. We are teaching ourselves. We then teach others, who eventually return to teach us. This is a rapidly expanding spiral gathering more information and participants with each revolution. Its great.

We are receiving calls from Amiga owners with new and different applications every day. They are extending the Amiga and its software which, in the process, secures the Amiga's acceptance.

However, no one programs the Amiga just to program. They are training the machine to perform a task. It might be something important or trivial, but the machine is just the tool to get the job done.

Education and the Amiga
This is the place for the Amiga and other computers in our society. Not objects in themselves, but the tools to reach a goal. In most of our educational institutions, the idea of the computer is turned the other way. The computer is an item that must be learned.

The term "computer literacy" is odd. No school system spends thousands of dollars to teach students the value and use of books. The students are taught to read by a slow process of indoctrination. They learn reading and books from an association with them in the process of discovering various other subjects.

It has been a long time dream of mine to see educational software follow the same paths. I long to see the scholastic publishers provide software that follows, explains, and clarifies their line of texts. If you bring software into a school to show children art and science, they will discover the computer as a tool.

With the Amiga, we are blessed with the facets of a computer that best matches a child's heart and imagination (it matches mine). With the graphic capabilities, sound and animation possibilities, the Amiga should become the next great educational computer. All it requires is intelligent software.

This software will need to be produced by educational and software professionals. However, this approach can produce the best software. This is a fantastic opportunity.

If you are a student who needs word processing software such as Scribble! or a parent searching for a math instruction program such as Discovery, this issue is for you. However, if you are the die hard "hacker" (in the true sense of the word), we have included plenty to keep you busy.

Don Hicks
Managing Editor

New Amiga Products From The Developers of Amiga C.

Amiga C Compiler—$149.95
Everything you need to develop programs on the Amiga, including a full set of libraries, header files, an object module disassembler, and sample C programs.

Unicalc—$79.95 A complete spread sheet package for Amiga, with the powerful features made popular by programs such as VisiCalc, SuperCalc, and Lotus 1-2-3. Unicalc provides many display options and generates printed reports in a variety of formats and print image files. Supports 8192 rows of 256 columns, and includes complete on-line help.

Lattice MacLibrary—$100.00
The Lattice MacLibrary is a collection of more than sixty C functions enabling you to rapidly convert your Macintosh programs to run on the Amiga. this allows you to quickly and efficiently take advantage of the powerful capabilities of the Amiga.

Lattice Make Utility—$125.00
Automated product generation utility for Amiga, similar to UNIX Make, LMK rebuilds complex programs with a single command. Specify the relationships of the pieces, and automatically rebuild your system the same way every time.

Text Utilities—$75.00 Eight software tools for managing text files. *GREP* searches for specified character strings; *DIFF* compares files; *EXTRACT* creates a list of files to be extracted from the current directory; *BUILD* creates new files from a batch list; *WC* displays a character count and a checksum of a specified file; *ED* is a line editor which utilizes output from other Text Utilities; *SPLAT* is a search and replace function; and *FILES* lists, copies, erases or removes files or entire directory structures.

Lattice Screen Editor (LSE)—$100.00 Fast, flexible and easy to learn editor designed specifically for programmers. LSE's multi-window environment provides the editor functions such as block moves, pattern searches, and "cut and paste". Plus programmer features such as an error tracking mode and three assembly language input modes.

OTHER AMIGA PRODUCTS AVAILABLE FROM LATTICE:

Panel: Screen Layout Utilities—$195.00
Cross Compiler:
MS-DOS to Amiga C—$250.00
dBC III:
library of data base functions—$150.00
Cross Reference Generator—$45.00

With Lattice products you get *Lattice Service* including telephone support, notice of new products and enhancements, and a money-back guarantee. Corporate license agreements available.

Lattice

Phone (312) 858-7950 TWX 910-291-2190

INTERNATIONAL SALES OFFICES:

Benelux: De Vooght. Phone (32)-2-720-91-28. England: Roundhill. Phone (0672) 54675
Japan: Lifeboat Inc. Phone (03) 293-4711 France: SFL. Phone (1) 46-66-11-55

Amazing Mail:

TO ALL WOULD-BE TURBO PASCAL PROGRAMMERS-

Wouldn't it be nice if there was a great Pascal compiler for the Amiga such as Borland's Turbo Pascal? As a programmer versed heavily in Pascal, I have been eagerly awaiting its release. Now it seems that Borland is going to deny us our Turbo Pascal after it had been promised to us back in January. A letter writing campaign may be all that is needed to show Borland that enough of us care and would like to have a Turbo Pascal compiler for the Amiga. So please write to:

Phillippe Kahn, President
Borland International
632 Kearny Street
San Francisco, CA 94108

The future of the Amiga and its support is now in the hands of us - THE USERS! Thank you and good luck.

Craig A Speigel
Plainsboro, NJ

Dear Mr. Speigel,
There are a great many reasons why a company may decide to first advertise a product and then not deliver. Due to the long lead times for most (other?) magazines, advertisers must have ad copy ready months in advance of the date they will run. The advertisers sometimes over anticipate both their capabilities and the market's.

It is unfortunate, when we see a large corporation, such as Borland, first announce their product and then drop it. They have successfuly stunted the growth of Pascal compilers for the Amiga; intentionally or not. With all due respect, we must ask "Are we seeing vaporware, or smoke?" D.H.

Dear Amazing:
I anxiously await the next issue of AC ..... regardless of when its published, its worth the wait. Sincerely,

Jerry Anderson
Kenner, LA

As you probably have noticed, Amazing Computing™ is now issued by number instead of by month. Our efforts to produce the magazine have continued to yield a 5 week production schedule. We believe in producing a quality product and if it takes us longer, we apologize. However, we are working harder than ever and we trust you understand and share our values. D.H.

Dear Mr Hicks:
Having MS-DOS experience, I found AmigaDOS reminding me of an Infocom adventure: Frustration one minute, exhilaration the next. (I hope Infocom is getting royalties for version 1.1 of AmigaDOS.) You, your staff and contributing authors have helped me tolerate the growing pains and envision Amiga's potential. I have two questions I hope you can help me on:

Downloading: IBM-XT vs Amiga
Repeated "time Trials" (to minimize such factors as line noise & a BBS's user load) at 1200b yield a download rate for an IBM-XT using Crosstalk XVI of about 50 xmodem blocks per minute. The best my Amiga can do using Online! is about 32 xmodem blocks per minute. Is it the DOS, software, modem, computer or a combination of such factors that could account for the Amiga coming in a distant second (I thought the Amiga had XT-type speed if you use ram: disk.)

"How to Sink a Sub"
I sometimes run into a "not deleted - object in use" error when trying to delete an EMPTY sub-directory. When I encounter this "error" - which is only some of the time - the only solution I've found is to re-boot. Needless to say, this is annoying me beyond amazement to the stage of bewilderment. If this question could be better answered by Infocom, please let me know.

Sincerely,
Larry Pahlman
Amazed User
Ft Wayne, IN

Larry:
1200 baud means 120 characters per second, or 7200 characters a minute, if you were transmitting non-stop. 50 Xmodem blocks a minute is about 6500 characters a minute, or about 110 characters a second. 32 Xmodem blocks a minute is slow, about 4200 characters a minute.

You are right, several factors could introduce this discrepancy. One is the transmission medium. A transfer from a bulletin board system in your city will happen at near maximum speeds. A transfer from a commercial service, such as Compuserve or People Link, will take place much more slowly, since the bits have so much farther to travel.

Another is the disk speed, and whether your terminal program buffers its transfers. If your program writes each Xmodem block to disk as it is received, this will retard the transfer. With Online!, it is best to transfer and receive files from the RAM: disk, since it does not buffer the incoming data.

There are many public domain terminal programs available for the Amiga. Often, the author has customized the program for a specific purpose, such as fast file transfers, or fast screen writes. Perhaps you should look at some of these programs. Some are faster than Online!. Keep in mind that MicroSystems Software has discussed an upgrade to Online!. In fact, they hired a former public-domain terminal program programmer, Michael Mournier, to assist them!

I've seen the 'object in use' message before, in similar circumstances. This is a genuine bug. J.F.

Dear Editor:
In the July 1986 issue Mr Rae (AmigaNotes) was complaining about not being able to set Preferences in Music Studio. Well, there is an easy way to do this. I must confess that I am not the originator of this information, I saw it in another magazine. Here is how to do it:

Boot with Kickstart.

Insert Workbench into the computer's disk drive, and open Workbench. Open the System drawer, and then open the Command Line Interpreter (CLI). Eject the Workbench disk, and insert the write-enabled Music Studio disk. Type the following (except the ">" prompt):

>preferences <press RETURN>

This will load Music Studio's Preferences. After changing them, click on "SAVE" to store them back on the disk. Next type:

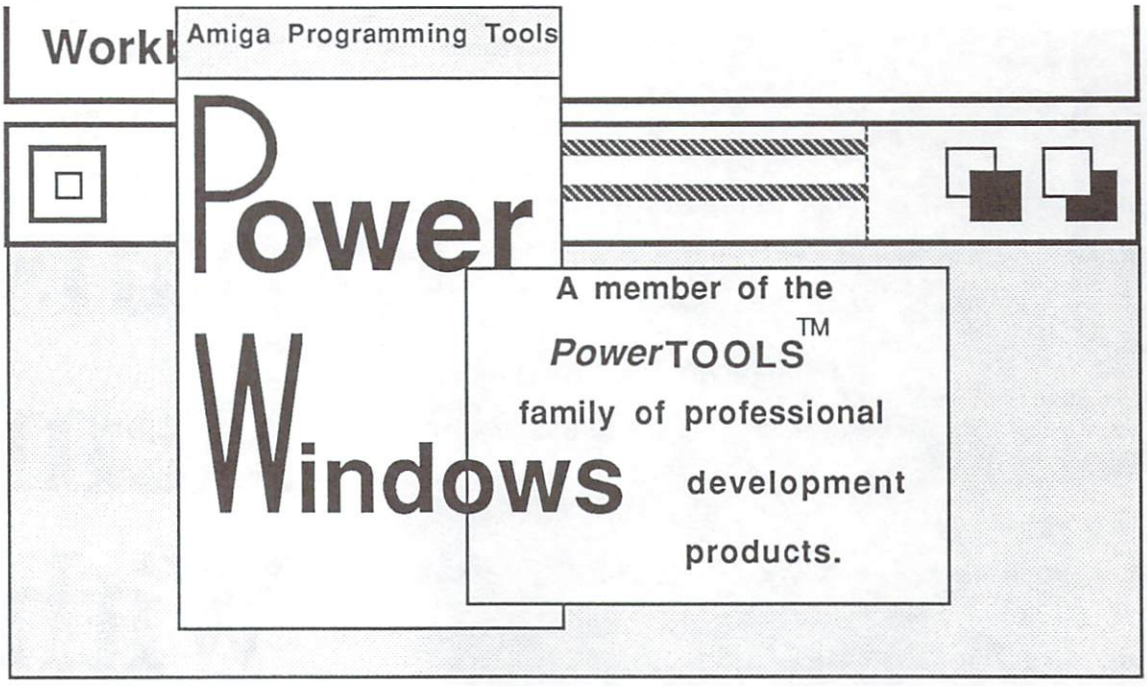
>endcli <pressRETURN>

If you have an external disk drive, then Music Studio may be inserted there and Workbench left in the internal drive. In this case the CLI command would be:

>df1:preferences <press RETURN>

Sincerely,
Allen Fincher
Suffolk, VA

•AC•



The **first** interactive Amiga program design tool, *PowerWindows*™ lets you design fantastic looking windows, menus and gadgets in minutes instead of hours or days! You show this incredible program what you want and it does the rest, generating C or 68000 assembler source code for you to include in your own programs. *PowerWindows* is a structure generator for a machine that **thrives** on structures. With this software package, written in assembler, you can:

Pick the exact size and position for your windows **visually.** No more "wait to see what it looks like"; *PowerWindows* knows where your window is and everything else about it!

Design professional looking menus. Add menus, move menus, or delete menus, whatever you want to do with text menus, our program keeps track of them and writes source code letting you duplicate them exactly with simple operating system calls.

Create your own string, integer and boolean gadgets and position them anywhere in your window. *PowerWindows* keeps them from colliding and remembers the type, location and text contents of each one for writing those complex gadget structures.

Best of all, you can keep your designs in a format that can be re-edited, letting you create your favorite type of windows and customize them for each program you write.

Order Form

Price for *PowerWindows* is $89.95, plus $3.50 for shipping and handling. Texas residents please add 6.125% sales tax to total price.

Name _____

Address _____

City _____ **State** _____ **Zip** _____

Products ordered _____

Payment method: _____ MC/Visa _____ Check _____ Money Order AC

Card Number _____ Expiration Date _____

Name on card _____ Signature _____

Enter total enclosed: _____

INOVATRONICS, INC.

11311 Stemmons Frwy., Suite 7 Dallas, TX 75229 214/241-9515

Expansion Memory Without The Wait.

Introducing *Alegra*: The Amiga™ Memory Expansion Unit from Access Associates.

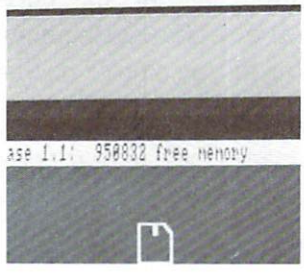
512 K now.

Now you can add 512 K bytes of external memory to your Amiga. In the smallest package available, a footprint only ¾″-wide. And Alegra's no-wait-state design lets your Amiga operate at its intended speed. No delays. With Alegra you get the benefit of fast memory at a surprisingly economical price. AND, BEST OF ALL, IT'S AVAILABLE NOW.

Upgradeable to 2 MB later.

If you'll need 2 MB of memory in the future, Alegra is still the right choice now. Our 2 megabyte upgrade (using 1 megabit DRAMs) will give you the memory you need in the same compact package.

Ask for Alegra at your quality Amiga dealer.



Total system memory is approximately 1 meg with the addition of our 512 K Alegra (depending on specific hardware configurations).

ACCESS ASSOCIATES

491 Aldo Avenue
Santa Clara, CA 95054-2303
408-727-8520

Alegra features a 90 day parts and labor warranty against manufacturing defects.

™Amiga is a trademark of Commodore Amiga, Inc.

The University Amiga

In each department, students, faculty and staff report their Amiga is useful in studying, teaching and research.

By Geoff Gamble
Washington State University

SOUND! ! GRAPHICS! ! These were the strengths advertised and if anything they were understated. It only took a few of the early reviews of the new Amiga to convince many of us at Washington State University that this was our computer. The list of Amiga users grows every month as our students and colleagues see what our machines can do and begin planning their purchases.

Amiga users can be found in a wide variety of academic departments on our campus including Music, Genetics, Philosophy, Microbiology, Electrical and Computer Engineering and Anthropology. In each department, students, faculty and staff report their Amiga is useful in studying, teaching and research.

Music
The sound potential of the Amiga is being explored by several faculty and students in the Music department. One student, just completing her graduate degree, has written her thesis about electronic music and is using an Amiga with a Rhodes Chroma synthesizer and a J.L. Cooper Midi retrofit to put together some dazzling "sound and light" shows. She has also developed a series of "slides", using Deluxe Paint, to lecture on the strengths and weaknesses of analog and digital synthesizers.

One of our Music faculty is developing a "music unit" suitable for elementary schools (K-5) to teach concepts of music composition, synthesized music and which allows children to explore music through the use of the Amiga. She is also developing the "teacher-training" components that will be used by the University to assist teachers learning how to use these computers in their classrooms.

Graphic DNA Study
The graphics capability of the Amiga was quickly put to an educational use by a doctoral candidate in molecular biology. Trying to find an effective method of showing how DNA is spliced to make genes, how the genes are transcribed to make RNA and how the RNA is used by ribosomes to make proteins, he used Aegis Animator to draw pictures of the relevant sequences, and now can explain this complex process with the aid of animations.

Building up a data base of still and animated pictures and other materials necessary for his lectures, the Amiga is connected through the NTSC port to the classroom television monitors and he calls up graphics, data, graphs and animation to support his talks. The mouse can then be used to draw molecules, genes or whatever he needs--a sort of high tech overhead projector with infinitely more utility.

Art OnThe Amiga
The graphics capability of the Amiga has not escaped the notice of the art world in our area. One local artist, who has shown his computer art regionally, worked with an Apple for the past eight years. His initial programs were art generators, a non-interactive mode in which the computer generates the art forms. During the past two years he added a digitizer and quickly saw the interactive art possibilities.

By that time, his Apple had been pushed to its limit and the Amiga, with Deluxe Paint, seemed perfect. He has been porting his Apple programs and graphics files to the Amiga and has added Digiview to his hardware. He now grabs an image with Digiview and then modifies it with his custom graphics programs, producing unique computer art images. One of his custom graphics programs prints out image data, giving color information pixel by pixel.

A natural extension of this work will be art tiles, where his Amiga will generate information about the number and location of each color tile. He and his wife, who is a member of the Art department at WSU, are also doing silk screens on the Amiga. The computer prints out the various stencils needed for each color, these print outs are then converted to transparencies and then applied directly to the silk screen.

Amiga As A Technical Tool
On the more technical side of things, one of our computer systems programmers is working to develop computer aided software design on the Amiga. The Department of Electrical and Computer Engineering has purchased twelve Amigas to be used in the classroom. The first students to use them will be in a junior level course on computer system design. The students will be required to explore, design and implementation on the 68000 cpu and will each complete a specific interface project before the end of the term.

Language

My own specialty is linguistics and I study the languages of western North American. Since most native American languages are unwritten, it is necessary to use a phonetic alphabet. I need a computer that will allow me to use the International Phonetic Alphabet (IPA) in my data base and word processing. The Amiga is perfect.

Software is still lacking, but the machine will do everything I need. I have used computers in my research for several years and have grown accustomed to doing a lot of my own programming to get standard and phonetic characters to print properly together and also be displayed on the screen. I have finished an IPA font for the Amiga and hope that a good word processor which allows various fonts will be available soon.

I also plan on using the Amiga for a graduate seminar in language theory and artificial intelligence that is scheduled for the spring term. The students will be using the twelve Amigas recently purchased by the Electrical and Computer Engineering Departments.

Normal Use

Standard applications of the Amiga are also seen here at WSU. Most all of our Amiga users do word processing and most of us are waiting for that special software that will allow us to do footnoting, use different fonts, or whatever else we could do on our other computers.

With more and more software being marketed each month, it seems likely that many of our software wishes will be granted. All the Amiga owners here are pleased with the machine and continue to find creative ways to use it in their teaching and research.

•AC•

Amazing Reviews...

MicroEd

Thorwald Esbensen, one man making a difference.

MicroEd's Torry Esbensen
Amiga's Educational factory

No study of educational projects on the Amiga is complete without at least a mention of Tory Esbensen and MicroEd.

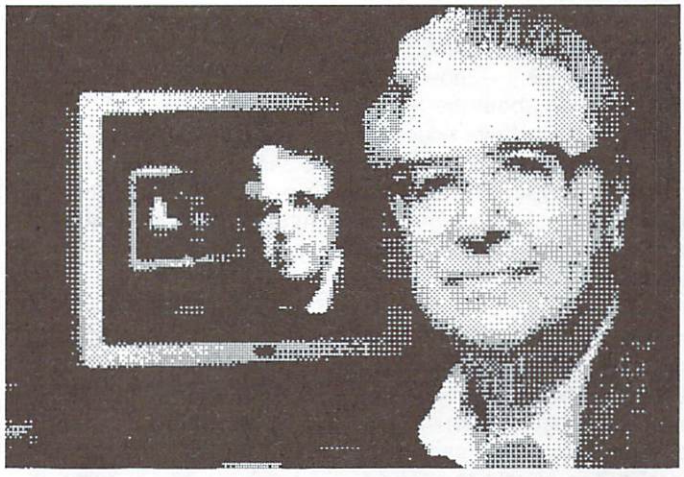
Thorwald (he signs his name Torry) Esbensen is a retired teacher and the current President of MicroEd in Eden Prairie, Minnesota. In the late Sixties and early Seventies, Mr. Esbensen was one of the first teachers to embrace the computer as a special teaching tool. In a state known for its progressive work toward computers and education, he found a tool that could be used for a more personal approach to education. However, he was also facing retirement.

Now at Sixty Three, Thorwald Esbensen is running MicroEd, a software company with no less than 22 programs for the Amiga, most are multiple disk sets. He has programs for history, spelling, reading, punctuation, and word skills.

MicroEd has introduced a series of historical programs that trace the evolution of the United States with paintings diagrams and photographs reproduced in the software with DigiView. The realism and impact of this technique must be seen to be believed. (See the review by Bob Frizelle, following, of MicroEd's The Lewis and Clark Expedition.)

Torry believes the computer is an ideal avenue for thought and education. He is quite proud of the work he has been able to do on the Amiga. Torry has developed all of the programs in Amigabasic, but said he was searching for a compiler to add a little speed. With few exceptions, the programs examined at Amazing have run well.

There is an added quality to the MicroEd programs. As Amigabasic programs, they have the bonus of being adjustable. If the user is willing, the source code is available to add changes to the test questions.



Torry Esbensen and friend

MicroEd programs available

Beginning Reading 1	$29.95
Beginning Reading 2	$29.95
Beginning Reading 3	$29.95
Beginning Reading 4	$29.95
Spelling Level 2	$29.95
Spelling Level 3	$29.95
Spelling Level 4	$29.95
Spelling Level 5	$29.95
Spelling Level 6	$29.95
Capitalization	$29.95
Punctuation	$29.95
Social Studies Vocabulary	$29.95
Vocabulary Series	$49.95
Basic Grammar	$29.95
Word Demons	$29.95
The Spelling Detective Game	$39.95
The Lewis and Clark Expedition	$89.95
Across the Plains	$59.95
Fur Trade of the Great Lakes	$79.95
Making our Constitution	$79.95
The Transcontinental Railroad	$39.95
Introducing Maps	$59.95

Amazing Reviews...

MicroEd, The Lewis and Clark Expedition

.....fifty digitized pictures showing just some of the settings for Lewis and Clark's great achievements

A Review
By Bob Frizelle

American History has always been a topic most people love to avoid. Let's face it --- how many people will sit down and have a conversation about the historical meaning of the Civil War? How many times have educators across the country heard the phrase, "Who cares about the pioneers journey across America? Why do I have to know names and dates of events that will have no signifcance in my future?" Indeed, teaching American History has presented Educators with a tremendous challenge.

To alleviate a little of the burden on teachers, MicoEd, Inc. has developed some innovative software for the Amiga. MicroEd has twenty two educational programs for the classroom. They range in category from "Beginning Reading" to subjects on American History. In each of the American History programs, MicroEd has sought to utilize the Amiga's amazing ability to digitize photographs and drawings found in the copyright- free books published by the U. S. National Park Service. It should be noted that in each of the programs in your Workbench Preference Drawer, the Text should be set at 60 and that mast Micro Ed programs require 512K and Amigabasic™.

The first program that I tried was the " The Lewis and Clark Expedition." This is a five part (5 disks) series that will provide the student with information about this great expedition. MicroEd features Learner - Controlled Instruction based on Mastery Learning. In layman's talk, this means you can decide how the material may be studied and when you are ready to be tested on this mateial.

You are presented with a screen of detailed instructions. For a student, the length of the instructions may become confusing, for example:

"A lesson is completed only when a PERFECT SCORE has been achieved on the test for that lesson. Each time the test option is chosen, twenty questions are randomly selected by the computer. If a question is missed, the test stops. From 1 to 99 points are then lost according to a Random Number Box. You get a number from the Box by pressing the RETURN key as directed.

The Mastery Test may be taken at any time, and be repeated as often as necessary. You start each lesson with 500 points. Your goal is to complete the lesson before you run out of points."

I don't think a student is going to be highly motivated by the fact that he or she must achieve a Perfect Score. Even though the test may be repeated as often as necessary, I think the teacher or student should be allowed to set a reasonable percentage of correct answers to be completed, (i.e. 85 percent or 90 percent). In this way the student will then receive some positive re-inforcement. Later, the teacher or student can increase the percentage until the Perfect Score is achieved.

Once the instructions are completed, you are presented with a Main Menu Screen. Here, you have three choices to choose from ;

A. See Information Options
B. Take the Mastery Test
C. Stop the Program.

Naturally, it is best to begin with the Information Options. You may then select one of the ten passsages about the Lewis and Clark Expedition. It is possible to select a different passage each time the Information Screen is presented. You do not have to proceed in any particular order.

Two questions are given after each passage. All answers must be typed with the Caps Lock key turned On. If it is not, your answers will be counted as wrong. However, no points will be lost here. The only time you will lose points will be on the Mastery Test. It should also be noted that Spelling counts, (the answer is wrong if it is misspelled), so take notes along the way! Also, don't press the Return key until the entire question is answered.

Beneath each passage will be a brief explanation of the full screen picture to follow. To view the picture, just press Return. The graphics here are quite interesting. The picture itself is actually taken from paintings and museum pieces. There is a total of fifty digitized pictures showing just some of the settings for Lewis and Clark's great achievements.

MicroEd's Lewis and Clark program is very good for the student who needs individualized instruction or immediate reinforcement. Because of the lengthy instructions and the content material, this program would be most beneficial to a higher grade level (grades 8 or higher). However, all students with an advanced reading level would benefit.

•AC•

Scribble! Version 2.0

Mail Merge, a Spelling Checker and a better run package, highlight this new version

There are two phrases in an educational setting that can pierce a student's heart and cause their blood to run cold, "Today we will have a pop quiz. " and "You will be required to do a paper that must be typed."

The first is usually spoken by the instructor right after you realize you have not read the daily assignment.

The second brings the realization of hours of research followed by notes and penciled drafts and hours of typing and retyping papers which will still be handed in fully painted with correction fluid. Most students spend more time either typing their paper or trying to find some one else to type it ("Would you, Mom, PLEASE?") then they do researching and correcting their grammar.

The introduction of *home* computers, has greatly reduced these problems. Students can use the word processing capabilities to produce their first draft and then rework the piece into their finished product electronically. The end result being, we hope, more time spent researching the paper and preparing ideas than painting out typos.

With the Amiga, we have seen a few word processing software programs with more ability than others (see Textcraft Plus A (Pre)Review by Joseph Lowery in Amazing Computing™ Vol.1 #6).

Micro-Systems Software, Inc. of Boca Raton, Florida was one of the first companies to publish a word processor utilizing some of the capabilities of the Amiga. The introduction of Scribble! allowed the Amiga user a choice from Commodore's Textcraft™ 0entry. With the updated Scribble! version 2.0, Micro-Systems Software has offered the Amiga user a wider choice and a better product.

Scribble! is now available with mail merge capabilities and a spelling checker ($99.95). The basic program has been improved and takes advantage of some of the tools available through Kickstart™ and Workbench™ 1.2 (neither have officially been released at the time of this review; we used beta 4 copies for our testing).

It is extremely nice when a requester box appears and you may type information without clicking a mouse in the text entry line. The menu and function boxes are also desellectable(?) by typing the first letter of the option (such as "O" for OK) to cancel the box and return to typing. Although these procedures worked on most of the boxes tested, the archive requester still required mouse interaction to proceed.

Scribble! will now produce Icons for your files which eases copying between disks and significantly speeds the startup sequence.

A close window gadget was added and up to four windows are supported. The standard memory default size per window is 16k. However, when loading a larger file, Scribble! will automatically compensate and is compatible with memory expansion card

Perhaps the nicest upgrade of all is it is free to all previous owners. A new disk is sent to all preregistered owners with spelling and merge capabilities as well as documentation of the new features.

Scribble! doesn't just "mouse around".
Although a considerable amount of work has been put into Scribble! to allow more interaction with menues and the mouse, there are still key command procedures required. This is not all bad. Scribble! does offer a screen of key commands for instant recall, by pressing F1.

The right to a choice between a mouse and a keyboard is a welcome change. Once a typist gets accustomed to a certain keyboard and system, they sometimes find it awkward to search for the mouse in the middle of typing.

Scribble! has implemented a few of the more popular WordStar™ commands for the user that is familiar with the system and, of course, there are the hidden commands required for printer use.

Mail Merge
The ability to produce mailing lists and multiple copies of letters is a great addition to this program. (Let's face it, where would the vacation condominium sellers be with out the ability to send you form letters telling what great prizes you have waiting.) Most early users will say they have no need for this capability until they have it. It is like owning a microwave, you don't need it until it is in the kitchen, then you can't live with out it.

Mail merge capability can be used for everything from form letters to bank statements. The Scribble! mail merge will eventually be linkable with Micro-Systems Software's soon to be released database program (they have a desperate fear of vapor-ware, so they will not advertised the name until it is ready). For now, the user must produce his files and lists in text format and file them in .dat files.

Scribble!'s mail merge works entirely with Dot Commands. A list of Dot Commands is available by pressing F2. Using the commands, you can name your data file and establish your

data names.

The variables are then inserted where required by placing ampersand (&) characters in front and behind of the variable name (&COMPANY& for COMPANY). The letter appears as it will when typed and all the variables are clearly marked.

Spell Checking
Definitely the best feature of Scribble! is the built in spelling checker. Spell checking a document, your active window, a selected line, or just a word can be done entirely with a mouse. The menues and screens are the smoothest of all features in the program. For a traditionally bad speller, I found this to be an exciting addition to the program.

Additions can easily be made to the dictionary, there is no means to remove a miss spelled word erroneously added. However, this is a fault of most spell checkers on all machines.

Printers
One of the strong points with the Amiga is its ability to match with virtually any printer. This gives it an advantage over other systems that are confined to the computer manufacturer's few choices. However, there is a drawback when producing a piece of software such as Scribble!

Although Bold, Italic, and Underlined text is easily selectable, the user is confined to the normal print text of his printer. Artwork and special fonts can not be activated in this program. This is a word processor, not to be confused with page layout software. The files are produced with text in mind, not graphics.

Micro-Systems Software Inc.
Scribble! is produced by a long time supporter of the Amiga. Micro-Systems has developed Online!, Analyze!, and BBS-PC! as well as Scriibble! They have an active 800 number for support and sales. They manintain a BBS for customer qusetions and they are on Compuserve's Amigaforum continually for any problems.

Caveates
Scribble is not perfect. There are places for improvement. I would personally welcome a different style of menu other than the one Micro-Systems has adopted. If the user does not work with the software continually, it can take a few minutes to refamiliarize yourself with the procedure. However, they have used this menu system throughout their Amiga line and if you own one Micro-Systems piece of software, you know how to use another.

In Conclusion
Scribble! is a good word processor and with the addition of the data base program planned by Micro-Systems, will prove a boon to users. The built in spelling checker is a great plus (just do not count on it to find all of your mistakes). If the price seems steep, just think back to those days when you watched the hours flow by as you were hunched over a typewriter with a bottle of white fluid...

Scribble! $99.95 ,
Micro-Systems Software, Inc. 1-800-327-8724
4301-18 Oak Circle 1-305 391-5077
Boca Raton, FL 33431,

Amazing Writers!!!

Yes, we mean you! If you enjoy Amazing Computing and you are using your Amiga, you have completed one half of the qualifications of an Amazing Writer for Amazing Computing™.

We are interested in the tasks and joys you have experienced on the Amiga. We want to read the secrets you have unlocked. We want to experience your excitement and enthusiasm. If you own an Amiga, you have already qualified as an independent thinker, now use that ability to communicate your individual story or idea.

Amazing Computing ™ pages are filled with people who want to reach you with their thoughts. They explain a portion of the computer you both use and abuse, because they found it interesting.

If there is something in the Amiga family that interests you, chances are there are people who would enjoy hearing what you have to say. So don't sit around waiting for others to teach you what you have already learned by hours of trial and error, get excited and teach the rest of us.

If your idea or explanation is of interest to developers and hard core hackers, please send your thoughts and a request for writer's guide lines to: AMICUS Network Editor.

If you are more interested in general use of the Amiga and its products, please send your suggestions and ideas to: Editor, Amazing Computing™

But, either way post them to: PiM Publications Inc.
P.O. Box 869
Fall River, MA 02722

Please include a hard copy and an electronic copy of your article for review. In both instances, please include your name, address and phone number. We will return an answer as soon as our editors stop shouting about how great your idea is, and types a response.

Amazing Computing™: your resource to the Commodore Amiga

Computers in the Classroom

We are the ones to open those doors.....

by Bob Frizelle

Panic

Imagine yourself as an Elementary School Teacher for the past twenty-five years in the same community. You have taught hundreds of children and watched as they grew older, then brought their children into the school system. In some cases, former students are now current colleagues. Gone are the days of Dick, Jane, Spot, and the twenty-cent lunches!

Before your eyes, you see materialize the "New" concepts in education. The idea of the Open Classroom, Team Teaching, Free Lunches and Breakfasts nearly make you pray for retirement! However, you weathered the storm as some of the "New" concepts saw their way out the door. What a relief! Now we can get back to the basics. Then it happened! The Dawn Of The Age Of The Computer!

Again, you are faced with introducing a new instrument into the classroom. To add to your dilemma, the students may know more about this instrument than you do! As each year passes, computers become more and more predominant in our society and we see more of them appearing in the homes. They help families keep track of their financial affairs. In addition, they provid the children with an instrument, not only fun to use but, will benefit their future in education.

If you, as an educator, are not prepared to teach or have a computer in the classroom, then I suggest you get prepared! This is no fad! We are dealing with an instrument here to stay.

We are on the threshold of a whole new beginning! We are the ones to open those new doors for the students who sometimes have difficulty in learning new concepts. We can do this by presenting them with a new instrument of learning. The Computer!

Legislatures across the country are gradually making money available, through special grants, to the school systems for the purchase of computers, as well as the development of a solid computer curriculum. For those school systems fortunate enough to have an active Parent Teacher Orginization, fund raisers have been held to supplement the grants. In some cases, these fund raisers have even supplied the schools with most of their computers and software.

There are those teachers who would prefer not to see the computer in their classroom. I understand, however, let's be reasonable and remember the reason we chose to be teachers; to provide the students every available opportunity to better themselves. If the computer can do this, let it!

Granted, there are going to be scheduling problems . How can you be fair to each and every student? When will they get to use the computer? This is an area that each teacher will have to work out for themselves. I suggest you discuss this with the students. Find out what they expect! From there, develop a schedule comfortable for you and them. In this way, I don't think anyone will feel left out or disappointed.

Software

The software being offered today is exciting. The students are allowed to reinforce and review their weaknesses in subject areas on an individual basis. In addition, they are doing this in a fun way with no teacher hanging over their shoulder! If the software is available, they can choose the subject area they feel they need to reinforce. What better way to learn?

The software waiting to be written for the Amiga will be Amazing! No hype, just fact. There are more tools available on the Amiga then on any of the machines now used in classrooms. Amiga's tools (color, graphic ability, video synchronization, sound, speech, multitasking and more) give the Amiga an incredible advantage for educational programs. These tools bring a subject alive in a personal, one to one manner, that no other medium (except a highly talented and motivated teacher) can match.

Special Needs

I have only discussed what the computer can do in the regular classroom. Imagine what the Amiga can do to help the special needs students! With voice synthesis, learning disabled children with speech and physical difficulties will be able to communicate in a totally new way! Just think of what that will do for their egos?! Let's not forget the other special needs students. How can the computer benefit them? It can give them the sense of accomplishment that they may otherwise not feel. By manipulating a joystick or pressing the correct key, this student may then achieve a level of success that he or she has never achieved before. What more could a teacher ever ask?

In the future, schools can expect to see more of the Interactive-Tutorials. This is where the student can view anything, anywhere in the world using actual VCR footage or a video disk! Just think of the possibilities this presents to them? They will have the opportunity of not only reading and answering questions about a particular subject area, but they will get to see pictures of it! They will then have the opportunity to review this material at there own leisure. Just think of the endless possibilties this will present to those with short attention spans? They now have the chance to prove to themselves, as well as to their teachers, that they can do the work!

With the advent of Gen-Lock on the Amiga, we will be able to test and caption material in various ways, which will continue to add variety and aditional facets to the subject material.

Presentation

There are two ways to present the computer to the students;

A.) Each classroom will have its own computer

B.) A laboratory setting where the students are assigned one hour of instruction dealing entirely with computing.

Why not combine the two of these? The student would then learn certain skills in the the Lab, and then be able to reinforce those skills, on an individual basis, in the classroom. Let's face it! Computers are here to stay! They are the future! It is our responsibility, as educators, to prepare them for that future!

•AC•

Amazing Reviews...

Two for Study

Some Work and Some Fun for the Amiga Young

Discovery,
"An Interactive Space Adventure"

"A ride on a Starship could be everyone's dream..."

Reviewed by
Robert L. Frizelle

To crash land on a remote, inhospitable asteroid would be everyone's nightmare. While surveying the damage, you discover that you alone are the sole survivor from your crew of four. The cargo of alien animal specimens have escaped from their containers and are lurking loose aboard the ship. Finally, you discover twelve fuel crystals have been dislodged and must be replaced.

Sound like fun? If it does, try "Discovery" from Micro Illusions.

"Discovery" is described by Micro Illusions as, "An Interactive, Educational Space Adventure" with versions in both Mathematics and Spelling. (We have yet to see the Spelling version). Children will not be disappointed. What is required is a 512k Amiga, a disk drive, a joystick, and a little mental agility (pencil and paper may not be a bad idea either).

The goal is quite simple, collect the twelve fuel crystals and restore the Starship to flight without being contaminated by the animal specimens. In the process you will want to achieve the highest score possible by solving the math problems.

"Discovery" has an AMAZING introduction that makes you feel as though you are drifting through the universe. It features, per the package, "Spectacular computer graphics, outer-spacial music, a starship that talks back and variable levels of challenge".

At the beginning you are introduced to the crew:
Scott, a male earthling
Katy, a female earthling
Mek, a robot
Lotar, a citizen of the galaxy.

You choose who you would like to be with your joystick. Also, you will choose the level of difficulty. You set the challenge levels (1 through 7 for math solutions). In addition, there are also 7 levels of addition problems, 7 levels of subtraction problems, 2 levels of multiplication problems, 2 levels of division problems and one with mixed problems to choose from. The difficulty increases with each level.

My initial disappointment with "Discovery" was how to choose a crew member. This version came with only one set of Instructions explaining the challenges. The instructions on how to get started were missing from our review copy. However, a quick call to Micro Illusions resolved this. The Joystick connects to the second mouse port.

Now, you are ready to begin your adventure. The monitor displays the interior of the Starship and your character with an outline of the ship below. A small yellow dot marks your location. Also displayed is the number of crystals collected, your strength (you begin with 20 strength points) and your score.

To increase your score, you must solve each math problem presented at the security doors. To increase your strength, you must collect the crystals. You lose strength (5 points) each time you come in contact with one of the Alien Creatures (included among them are Veerus, the Meba, and the Cellot).

Solving the various math problems may sometimes become confusing. For example:

Addition: - You must work from right to left. Add each column of digits separately and enter the correct number under each column. Next, enter the remainder. It will then appear at the top of the column to the left. NOTE: If the remainder is zero it must also be entered and carried over.

I think children will have a little difficulty with putting a zero at the top of the next column. That is not taught in the classroom. Also, there is no way to correct a wrong answer.

Again from the instructions:

Subtraction: To find the difference work from right to left subtracting one column at a time. If the digit below is larger than the digit on top press "B" to Borrow from the column on the left and enter the remainder above it. You may then return to the column on the right and subtract.

I feel children will have difficulty with pressing the "B" key to Borrow and entering the remainder. They very well could get confused and enter the wrong number. Also, there is no way to correct a wrong answer.

The Multiplication and Division levels will initially get the children confused with how to enter their responses. With addition and subtraction levels, you enter responses from right to left. Here, you enter responses from left to right. Again, I think the children will become confused in answering the problems. What AMAZED me also was the Backspace key could be used for correcting wrong responses. The graphics displayed before you are very good.

The manipulation of the joystick will present an even greater challenge in avoiding the alien specimens. The audio can sometimes become repetitious (to be honest after 30 minutes the music will drive any teacher crazy) and can be turned off by entering the letter 'M'. Although the sound has some use to the player, I would recommend a set of headphones if this were to be used in the classroom.

"Discovery" does present a challenge as you encounter each security door. If you respond incorrectly, the door will not open. Respond correctly and you are free to move to another compartment. Respond incorrectly and you are stuck! The correct answer will then be displayed next to yours. You must go backward one square and then forward to be given another chance to answer correctly.

From an educational point of view I don't feel "Discovery" will offer an exciting challenge throughout the game. From a child's point of view "Discovery" will offer an exciting challenge through a Starship, adept skill at joystick control and intellectual satisfaction that he/she achieved what they set out to do!

Discovery "Math"
$39.95
MicroIllusions
P.O. Box 3475
Granada Hills, CA 91344

The Talking Coloring Book

There is no mess to clean up , no walls to wash....

Reviewed by
Robert L. Frizelle

Children Love to color. From the moment they are old enough to stand (or even younger), they love to scribble something on anything. Parents and teachers, you know what I mean! How many walls have crayon marks on them? How many chairs or couches have "mysteriously" appeared with crayon, ink, or marker stains after a rainy day?

If you are a victim of these symptoms, then you should try "The Talking Coloring Book" by JMH Software of Minnesota Inc. It is *amazing* for young children (as well as you adults that sometimes like to live a second childhood)! There is no mess to clean up, no walls to wash and no screaming at the kids. What more could you ask for? What you will need to get started is your Amiga and your mouse. Simple enough (don't forget your imagination)!

After you insert your disk, you will be presented with 4 options. Let me briefly explain each one.

Option A.

DEMONSTRATION--- The computer will speak a color word, then show that word and color on the screen. This is very good for color identification as well as reinforcing the color words.

Option B.

PRACTICE--- Here , the computer will challenge young (and Special Needs) children to identify each of 9 different colors. A little co-ordination on the child's part using the mouse pointer to the correct color and pressing the left button produces the correct results. I think children will get great satisfaction at knowing they have responded correctly to the computer. What happens with a wrong response? Try again!

Option C.

COLOR--- (This is the option everyone will like)! You get to choose a picture, (an airplane, clown ,Teddybear, etc.) and color it brilliantly with your own imagination. How? Choose the color you desire with the mouse (press the left button), go to the area you want to color and press the left button again. Bingo---the computer fills in the desired area with your color.

Option D.

DRAW Again, I think everyone will like this option! Where else can you be another Picasso? In this option, you draw anything you wish, change it and store it on the disk! Later you can go back and color each picture. If you do not like it, delete it!

The Talking Coloring Book" offers children of all ages an introduction into the world of coloring and interaction with the computer. It is very easy and fun to use! If you have young children who like to color, a Primary School Teacher or a Special Needs Teacher (with an Amiga in your classroom), then I would seriously recommend "The Talking Coloring Book".

The Talking Coloring Book
$29.95
JMH Software of Minnesota Inc.
7200 Hemlock Lane
Maple Grove, MN 55369

•AC•

Amazing Reviews...

True BASIC

*Finally a **standard** for the BASIC Language!*

Reviewed by
Brad Grier

In the early 1960s, computer languages were cryptic and meaningless to anyone who hadn't studied them for years. As more students became interested in learning about and using these computers, a need arose for an easier programming language.

Recognizing that need, a group of talented undergraduate students and their professors developed BASIC, the Beginners All-purpose Symbolic Instruction Code, one of the first interactive computing languages. BASIC is the most popular computer language in use today.

Early microcomputer users recognized BASIC as a perfect language for their machines. It was an easy to learn language, and that helped to sell computers.

As BASIC was used in different computers, the language itself began to change. Software designers had to create new functions to take advantage of slow, mass storage devices like cassette recorders. The language had to be pared down so that it would fit in the computer's memory, often to a size of only 8k. Soon the dialects of BASIC were quite different. Microsoft BASIC is one of the most popular dialects.

But the incompatibility between the language and the hardware still exists. If a program uses a 200 pixel-wide graphics screen on one computer, it cannot run on a higher resolution computer, without modification.

True BASIC

Because of the many different dialects of BASIC, the American National Standards Institute set up a committee that would examine the features and benefits of all BASIC dialects. From the best of these, they would created a standard for the BASIC language. This standard would bring BASIC up to the state-of-the-art level in computer languages.

One of the members of the committee is Tom Kurtz. He, with the help of John Kemeny and some undergraduate students designed the original BASIC language. They also designed the True BASIC Language system.

True BASIC was designed with these ANSI standards in mind. It is an ideal language for the Amiga. It allows older constructs like line numbers, GOSUB's, and GOTO's, but also has some of the more advanced structures like, SELECT CASE, DO-LOOP, DO-WHILE, and DO-UNTIL.

True BASIC is portable. Any program written in True BASIC on an Amiga, when moved to a Macintosh or IBM, will run without any modification of program code - including graphic-oriented programs.

The Editor

An editor helps you enter your program. Both True BASIC and Amiga BASIC have full screen editors, but the True BASIC editor is faster, and easier to use.

True Basic's editor is chock full of gadgets. Most notable is the Error Message window. It is a small window along the bottom of the edit window, just large enough for a line of text.

This window is normally empty. When it encounters an error in your program, it flashes orange, and displays an error message. The cursor then goes to the line that caused the error. It can display up to five error messages, but only one at a time. You must click in the window to bring up the next error message.

The editor contains the usual functions you would expect, scroll bars and arrows, cut and paste, global search and replace, etc. But it also has some commands that need a little bit of explanation.

Include
Reads a disk file into the program you are currently editing.

Keep
Discard all lines but the program block you have specified.

Edit
Limits the scope of the editing window to the block of lines you have defined.

Move to block
Move cursor to a sub-routine of block you have defined with a name.

Compile
Breaks the True BASIC program down to a binary file, which will reduce program size and execution time.

The editor has all the essential features of a basic word processer. In fact, some of the text files on the disk were written using the editor.

Windows

True BASIC has 3 windows, Command, Source, and Output. With these, it is easy to write, execute and debug programs.

The Source window is where the programming takes place. The full screen editor, and menus reside here.

The Command window is where you can enter basic commands for system status and debugging purposes. If you wanted to know what value was assigned to the variable X, open the command window and type "Print X" and hit <RETURN>. The value of X would be printed below.

Do you want a list of files on disk? Open the Command window and type "Files" and enter <RETURN>. True BASIC will display all the files in the current directory. Any True BASIC command can be entered in the Command window.

The Output window is where any output generated by your program will appear. Normally this window is closed until you run a True BASIC program. Then it is opened and any text or graphic output is displayed. You can open it manually, and keep it displayed while you edit your program in the Source window.

True BASIC will support up to 10 windows under program control on screen at a time. These windows may contain text or graphics or a mix of both. Only one window may be 'Active' at a time, but your program can decide which window to write to at what time.

Programming in True BASIC

If you have ever programmed in Amiga Basic or Microsoft Basic, you will be suprised how easy the True BASIC language is to use. As a test of the language, I wrote the SpaceZap program - in an afternoon! The program incorporates sound, graphics and mouse input, in under 100 lines.

This program can run on ANY computer running True BASIC, while an Amiga Basic version would work only on an Amiga (For more information, see the sidebar 'SpaceZap' or 'Orbits'.)

To write the equivalent program in Amiga Basic, I would have had to use the Sprite editor to create a Sprite, then add many more lines of support code.

The language has many familar BASIC commands. It also incorporates many new ones. These are described below.

The MAT statement, when prefixed to a True BASIC command, will allow you to manipulate the data inside an array, without setting up a loop.

MAT READ - Like the BASIC READ command, except it reads all elements from a DATA statement into an array until the array is full, without a For/Next loop. Example:

```
!    Set up by reading months and days of week.
!
dim month$(12), weekday$(7)
mat read month$, weekday$

data January,February,March...(till the array is full)
data Monday,Tuesday...(till 2nd array is full)
end
```

MAT PRINT will print out all elements of an array. Example:

```
!    Print a simple little list.
!
dim a(10)

for i = 1 to 10      ! load the array inside this
   let a(i)=i        ! loop
next i

mat print a          ! print the entire array
end
```

PICTURE is a command that will allow you to define the method of drawing a picture as a function. You can then call it as a subroutine and pass variables to it to alter your graphic. Then by calling the Draw command, you can draw your picture. Example:

```
!    Polygon picture.
!
picture Polygon(sides)

   for i=0 to sides          ! run through vertex points
     let u=2*Pi*(i/sides)    ! find next vertex
     plot Cos(u),Sin(u);     ! plot next point in series
   next i
   plot                      ! turn off beam when done

end picture
set window-1,1,-1,1          ! define ploting/screen area

for n=3 to 10                ! polygons of 3-10 sides
     draw Polygon(n)         ! draw polygon of this order
next n
end
```

TRANSFORMS, or Picture Transformations are methods of altering an entire picture before drawing it. True BASIC includes five methods of transforming a picture:

Shift(a,b) - slides picture over. Every point plotted at point (x,y) will be moved to location (x+a,y+b).

Scale(a) - scales picture size. Every point plotted (x,y) will end up at location (x*a,y*a).

Scale(a,b) - same as Scale(a) except points plotted (x,y) will end up at (x*a,y*b).

Rotate(a) - rotate a picture (a) radians (or degrees) counter-clockwise about the window origin.

Shear(a) - lean all vertical lines in the picture clockwise by (a) radians or degrees. Points plotted (x,y) will move to location (x+y*tan(a),y).

Transformations may also be applied to arrays, making array manipulation easier.

ON-LINE HELP. True BASIC includes an extensive on-line Help facility. To use it, either enter 'Help<CR>' in the Command window, or hit the help key. You will be presented with a window and a requester. Scroll to the item you need help with and click twice (or type the first few letters of the command). A window will open up with text pertinent to the section that you need help with.

PLOT - You can plot any point on the screen by using this command, 'Plot x,y'. If you want to plot many points, you could use 'Plot Points: x1,y1;x2,y2;...'. If you want to draw lines between points, just add a final semicolon to the 'Plot x,y;' statement. That will leave the beam on. To fill an area enclosed by points, use the 'Plot Area: x1,y1;x2,y2;...' command.

The Plot command can be combined with the Window command to make quick and painless graphs. By adjusting the window size, you can create a screen with meaningful points.

For example, if you wanted to graph the number of cars manufactured in North America every 5 years, over the last 30 years, you could define the left edge of the screen as 1956, and the right edge as 1986. The top could be 100 million and the bottom could be 0. Then all you would have to do is to plug in the meaningful data, and watch the results appear when you run it.

The True BASIC system allows you to use subroutine libraries. This means that if you have an often-used routine in your programs, you can incorporate it by storing it in a library, and calling it from your program. [Ed. These libraries are not the same as the Amiga libraries discussed in other articles, they are specific to True BASIC.]

You can pass variables and values to and from the library routines, as if you had written them yourself. This greatly simplifies the programming process. You can write and debug several modules, then link them together in your program code to produce a finished program.

You will not be disappointed when you open your copy of the True BASIC Language system. Inside you will find two manuals, the User's Guide and the Reference Manual, a disk containing many example programs, and a reference card.

Both manuals are coil bound, and published by Addison Wesley, the same folks that brought out those excellent Amiga reference manuals.

The User's Guide is aimed at first-time BASIC users. This manual assumes that you have no previous knowledge of BASIC, and leads you through the programming language quickly. It explains each command and demonstrates each in example programs.

The manual is also a tutorial. You will be asked to edit and change some programs to examine the differences some parameters have on different commands. The User's Guide will leave the novice BASIC programmer feeling that they can easily write a program in True BASIC.

The Reference Manual is the source for specific information about any True BASIC command. Each chapter details a different aspect of True BASIC, from graphics to error handling. The chapter presents numerous examples referring to the individual function described.

The manual also describes advanced concepts such as packing routines, graphic manipulation, file manipulation, and compatibility between different BASIC dialects. The experienced BASIC programmer should be very happy with this manual.

True BASIC is a very neat package. It is quick, clean and bug free. I made a simple benchmark between True Basic and Amiga Basic. True BASIC ran twice as fast. The benchmark was a simple program to count to a million. Since it is possible to produce stand-alone programs with the Run Time Package, you can easily use this language as a development tool for commercial applications.

Currently, there are a number of external program libraries available. These include 3-D graphics, a PC BASIC converter to translates other BASIC dialects to True BASIC code, Communications Support, Sorting and Searching, Advanced String Library, and a few more.

The Amiga Developers Toolkit is due soon. This library includes routines for advanced graphics and sound functions, advanced animation routines, and access to all system functions.

True BASIC presents itself as competing with Microsoft BASIC and Borland Turbo Pascal. Their brochures compare each to True BASIC. The True BASIC program was by far the easiest to read and the shortest program listed. [Ed.: After all, it is an advertisement.]

I have included two short True BASIC programs in this review, to illustrate the features and simplicity of the language. In my personal opinion, this language should have been the one shipped originally with the Amiga from Commodore. I like it.

SpaceZap.tb

SpaceZap was written as a test to see if True BASIC would support intermediate level graphic routines. True BASIC does not currently support sprites and bobs but the True BASIC developer's toolkit will.

As it currently stands, good quality graphics are available in True BASIC. Using Plot Area commands, in conjunction with Picture statements, you can simulate sprites, as is done in this program. True BASIC's sound command is used to get the sound of the laser.

By writing this program, I discovered that complex programs are made less complex by using a more sophisticated programming language.

```
Listing of 'SpaceZap'

!       SpaceZap.TB             !
!       a shoot-em up           !
!       in space written        !
!       in True Basic.          !
!                               !
!       By Brad Grier           !
!                               !
!                               !
INPUT prompt "Skill level (0-5) ":skill  ! Get skill level
LET skill=skill/5
SET back "black"                    ! Set background to black
SET window -500,500,-500,500  ! Set origin to center screen
LET ox=100                    ! Initial ship x axis position
LET oy=100                    ! Initial ship y axis position
LET a=9                       ! Define Plot offset multiplier
LET shots=1
!
! Draw the ship
!
PLOT area:ox,oy+3*a;ox+2*a,oy-2*a;ox,oy;ox-2*a,oy-
2*a;ox,oy+3*a
                                                  ! Draw ship
BOX KEEP ox-3*a,ox+3*a,oy+3*a,oy-3*a in ship$
                                        ! Store ship shape in
CLEAR                                     ! string variable
DO                                  ! *** MAIN LOOP ***
   LET tx=0
   BOX SHOW ship$ at ox,oy          ! Draw ship
   BOX CLEAR ox,ox+6*a,oy+7*a,oy    ! Erase ship
   GET MOUSE: x,y,state ! Get mouse input for ship movement
   IF abs(ox+3*a)-10<0 then LET tx=tx+1     !--\
   IF abs(oy+3*a)-10<0 then LET tx=tx+1     !Plot x & y
   IF state>0 then LET tx=tx+1      ! Fire pressed? Add 1 to
                                    ! shot count
   IF state>0 then                  ! and make noise
      FOR ss=1 to 3
         SOUND 100,.03
         SOUND 500,.03
      NEXT ss
   END IF
   IF tx=3 then                     ! Hit anyone?
      LET score=score+1             ! if so, increase score
      LET ki=ki + 1                 ! and increment skill count
      LET ex=rnd*50                 ! Draw explosion
      SET color "red"
      FOR i=1 to ex
         IF i>(ex/2) then SET color "yellow"
         PLOT 0,0;rnd*100,rnd*100
         PLOT 0,0;-rnd*100,-rnd*100
         PLOT 0,0;-rnd*100,rnd*100
         PLOT 0,0;rnd*100,-rnd*100
      NEXT i
      CLEAR
      SET color "white"
      PRINT "Shots";shots           ! Display score
      PRINT "Kills";score           ! Display kills
      PRINT "Skill";skill*5         ! Display Skill level
      LET ox=(rnd*1000)-500         ! Get x pos for new ship
      LET oy=(rnd*1000)-500         ! Get y pos for new ship
   END IF
   IF x1<>x then LET mx=x+x1  ! Missed ship, compute x
   IF y1<>y then LET my=y+y1  ! Missed ship, compute  y
   LET ox=ox-mx/50
   LET oy=oy-my/50
   IF state>1 then
      GET MOUSE:N1,n2,null          ! Reset fire button
      SET color "red"          ! change colour of cross-hairs
      LET shots=shots+1        ! add one to the shot count
   END IF
   PLOT 0,0             ! Draw cross-hairs in current colour
   PLOT 50,0; 100,0           ! White if no shot fired and
   PLOT -50,0; -100,0               ! Red if shot fired.
   PLOT 0,50;0,100
   PLOT 0,-50;0,-100
   IF state>1 then SET color "white" ! Fired? change colour
                                     ! back to white
   IF ki>4 then             ! If you have more than 4 kills at
      LET skill=skill+.2 ! the skill level, increase
      LET ki=1           ! skill level by 1
   END IF
   LET x1=x
   LET y1=y
   LET ox=ox+(skill*(rnd*35))   ! Plot ship evasive manuvers
   LET oy=oy+(skill*(rnd*35))   ! based on skill level.
   LET ox=ox+(skill*(-rnd*35))  ! higher skill, the more
   LET oy=oy+(skill*(-rnd*35)) ! the ship will move around.
   IF oy>490 then LET oy=490
   IF oy<-490 then LET oy=-490
   IF ox>490 then LET ox=490
   IF oy<-490 then LET ox=-490
LOOP
END
```

Orbits.tb

'Orbits' is a little program that I have adapted to the Amiga. It serves to demonstrate the ease of conversion to True BASIC from other dialects of BASIC.

'Orbits' allows you to enter the gravities and positions of up to ten objects into the computer. 'Orbits' will then compute and display each object in relation to other objects.

This program was easy to convert. The math formulas were transferred intact, and the program logic is unchanged. The only alterations made were in the screen size, and the screen addressing, with the Window and Plot statements. These were adjusted to take advantage of a higher resolution screen, with more colors available.

```
Listing of 'Orbits'

! Orbits
!
! From the August 1980 Interface Age.
! Originally written for an Atari 400.
!
! Modified to True Basic August 20, 1986
!
! Set up the simulation
!
SET mode "high16"    ! Enter Hi-resolution graphics mode
SET window -500, 500, -500, 500
                     ! Set origin at center of screen and
SET color "blue"     ! screen width and height at 1100 points
SET back "black"     ! each.
DO while t < 2 or t > 10          ! Get number of objects
   PRINT "Number of orbiting objects (10 maximum)";
   INPUT t
LOOP
!
! Set up memory arrays
!
DIM G(10), X(10), Y(10), U(10), V(10), R(10)
! Change the 10's for more objects.
!
! Get values for all objects
!
FOR i = 1 to t
    PRINT "For orbiting object #";i     ! Get gravity
    PRINT "Gravity=";
    INPUT G(i)
          let R(i)= G(i)/5
    PRINT "X-Coordinate=";              ! Get initial X co-ord
    INPUT X(i)
    PRINT "Y-Coordinate=";              ! Get initial Y co-ord
    INPUT Y(i)
    PRINT "X-Speed=";                   ! Get X-speed
    INPUT U(i)
    LET U(i)=(U(i))/100
    PRINT "Y-Speed=";                   ! Get Y-speed
    INPUT V(i)
    LET V(i)=(V(i))/100
NEXT i
CLEAR
!
! Go on to compute and plot points
!
DO
   FOR i = 1 to t
       FOR j = 1 to t
           IF i <> j then
              LET X1=X(j)-X(i)
              LET Y1=Y(j)-Y(i)
              LET D2=X1*X1+Y1*Y1
              LET G1=G(j)/(D2*sqr(D2))
              LET U(i)=U(i)+G1*X1
              LET V(i)=V(i)+G1*Y1
           END IF
       NEXT j
       IF i = 1 or i = 9 then SET color "blue"
       IF i = 2 or i = 10 then SET color "white"
       IF i = 3 then SET color "red"
       IF i = 4 then SET color "yellow"
       IF i = 5 then SET color "green"
       IF i = 6 then SET color "cyan"
       IF i = 7 then SET color "magenta"
       IF i = 8 then SET color "brown"
       box circle X(i)-(R(i)/2),X(i)+(R(i)/2),Y(i)-
(R(i)/2),Y(i)+(R(i)/2)
       !PLOT X(i),Y(i);  ! Plot first position beam on
       LET Y(i)=Y(i)+V(i)
       LET X(i)=X(i)+U(i)
       box circle X(i)-(R(i)/2),X(i)+(R(i)/2),Y(i)-
(R(i)/2),Y(i)+(R(i)/2)
       !PLOT X(i),Y(i)    ! Plot second position beam off
   NEXT i
  ! CLEAR          ! Remove the "!" to erase trails
LOOP
END
```

•AC•

PUT YOUR AMIGA TO WORK

with

DATAMAT™

FULLY RELATIONAL DATABASE MANAGEMENT SYSTEM.

- **Now with images in IFF format, display with text/data/voice**
- **Quickly build applications without any program coding from simple phone/mailing list to research to organization-wide information management**
- **Self-running tutorials created automatically for personnel training**
- **Integrate with virtually all existing hardware systems Companion software with identical user-interface for MS DOS, XENIX, UNIX, VMS, and others available. Same application fits all hardware**

DATAMAT PARTIAL SPECIFICATIONS

Organization	Fully Menu-driven Relational Database Management System/Application Generator.	Number of data files per data base	Unlimited
Number of characters per field	1,024	Data types	13 includes Image in IFF Format
Number of fields per record	2,000	Global (System) Fields	40 user definable 9 special purpose
Number of characters per record	4,000	Field checks	Mandatory, Type, Initial value, Value within a specified range.
Number of records per file	4.3 billion	Password security	Field and data base levels
Multiple response	Supports multiple responses (up to an array of nine) for a single field.	Calculation capabilities	Full complement of 23 math and trigonometric functions and 13 logical operators. Automatic date and time calculations.
Number of Relations per data file (simultaneous R/W access)	10		

Data Entry - single entry to multiple files and records. **Import/Export** facility with data conversion/reorganization. **Forms Definition** - full screen editor with mini word processor. **Report Generation** - up to 66 lines x 132 columns, 6 level totaling with built in summary. **Sort/Search** - up to 26 selection criteria per query. **Mass Editing, Time Saver Audit** - stores all key strokes used in building application for automatic recreation. **Statistics and Graphics**- stepwise multiple regression, standard statistical tests and analysis; scatter plots, bar charts. Custom Applications Generator - batch/partial batch processing; user-defined menues; self-running demos.

Available through your Amiga dealers. Inquiries Welcome.

Dealers Contact: EJ Distributors (716) 876 1457
3170 Delaware Ave.
Kenmore, NY 14217

Transtime Technologies Corporation

797 Sheridan Drive, Tonawanda, New York 14150; Phone: (716) 874-2010

Using Your Printer with the Amiga

"...the Amiga offers four basic ways of sending output to a printer."

by Sheldon Leemon

Two of the Amiga's most outstanding assets are its versatility and its friendliness. But sometimes even such positive qualities as these can present problems to users who are used to dealing with other computer systems. A prime example is the methods that the Amiga provides for the user to send output to printers.

Most personal computers only allow the user to deal with the printer in a strictly prescribed fashion. The printer is handled as a one of two possible DOS devices, a parallel device if the printer is attached to the computer's Centronics parallel port, or a serial device if it is attached to the RS-232 modem port. In order to send characters to the printer you must open the device that corresponds to the method you've used to attach your printer to the computer, write the characters that you wish to send, and close the device. The characters that you send go directly to the printer, unchanged in any fashion. If you want to make the printer perform one of its special functions, like underlining or condensed print, you must send a printer-specific code directly to the printer. These codes are sometimes known as escape codes, since they generally start with the Escape character (ASCII character code 27, or CHR$(27) in BASIC). These codes vary from printer to printer, so the code that you use to invoke a special function on one printer will often not work for another printer.

But the Amiga offers four basic ways of sending output to a printer. The most common of these methods employs the AmigaDOS device known as PRT:. The PRT: device avoids the two biggest drawbacks presented by the old-style handling of printer output. First, it eliminates the need to specify to a program whether your printer is connected as a serial or a parallel device. Secondly, it makes it possible for the program to activate special features such as underlining without knowing the specific codes used by the printer that is hooked up to the computer.

When a program tries to access the PRT: device for the first time, two things happen. First, the software that handles the printer device checks the logical device DEVS: for a file called "System-Configuration" (the DEVS: device name is automatically assigned to the "Devs" directory on your startup disk). This file contains the settings that you have saved from the Preferences program. Your preference settings tell the PRT: device whether your printer is connected to the parallel or serial port, so it knows where to send its output.

The PRT: device also reads the "System-Configuration" to determine what type of printer you are using. This is determined by the name of the printer you have highlighted in the "Change Printer" screen of the Preferences program. After PRT: reads the name of the printer, it tries to load a "printer driver" file of the same name in the Printers sub-directory of the DEVS: logical device. If you print a listing of the DEVS:Printers directory, you will see that it contains files corresponding to each of the printers listed in Preferences. These "printer driver" files tell the PRT: device exactly what special features are available on your printer, and how to access them. They also contain programs which can convert information about screen graphics into graphics image data for those printers that support dot graphics or color graphics printing.

Once the PRT: device has read in the printer driver file for your printer, it "knows" how to activate your printers special features, so it is no longer necessary for a program to send that device the printer-specific codes for accessing those features. Instead, the program can (and must) send the "generic" Amiga codes for these features. There are currently 75 printer codes supported by the Amiga. A list of these codes, and Workbench printer drivers that support them, appears in Table 1.

For example, to activate your printer's underline feature from an Amiga BASIC program, you could send the Amiga "Underline on" code to PRT: (or LPT1: as it's also known in BASIC), regardless of what your printer's actual "start underline" codes are:

```
OPEN "PRT:" FOR OUTPUT AS #1
PRINT #1, CHR$(27)+"[4m";  'Underline On code'
CLOSE #1
```

You could also use the BASIC LPRINT command to accomplish the same thing:

```
LPRINT CHR$(0)+CHR$(27)+"[4m";
```

This method saves you the trouble of having to explicitly OPEN and CLOSE the PRT: device. The BASIC line printer commands, LPRINT and LLIST both send their output to the LPT1:BIN device, which is a BASIC variation of PRT:. However, you may have noticed that we preceded the Amiga escape code (CHR$(27)+"[4m") with the non-printing character

CHR$(0). For some reason, the BASIC LPT1:BIN devices requires that you send it a non-printing character such as CHR$(0) before sending the escape sequence. If you fail to send the CHR$(0), BASIC will not pass the subsequent Amiga escape codes correctly to the printer, and the special printer feature will not be activated.

You should be aware that when you use the PRT: device, your printer will only respond to the Amiga printer codes. Any attempt to send printer-specific codes directly to the printer through the PRT: device is doomed to failure.

The advantage of using the PRT: device over sending escape sequences directly to the printer should be obvious. A BASIC program that uses PRT: to access special printer features will work with any printer with a properly-installed printer driver, not just one particular printer. But note the qualification in that sentence, because it is an important one. In order to take advantage of the PRT: device, you must have a properly installed printer driver on the Workbench disk that you use to start the computer up.

There are two steps to installing a printer driver on your Workbench startup disk. First, you must set your printer defaults using the Preferences program, and save them. Preferences stores its information in a file called "System-Configuration" which is located in the "Devs" directory of your startup disk. The second step is to make sure that there is a printer driver file in the "Devs/Printers" directory that corresponds to the printer that you selected in Preferences.

The most important printer default that you must set from Preferences is the type of printer that you will be using. To specify your printer type, select the "Change Printers" screen from the main Preferences screen. A list of printers is displayed in the top right corner of the screen. The name of the printer that is currently selected is highlighted. To change the printer selection, click the mouse on the up arrow or down arrow next to the list. The standard Workbench disk contains support for a number of different printers; the Alphacom Alphapro 101, Brother HR-15XL, Diablo 630, Commodore CBM MPS 1000, Epson JX-80, Epson X-80 series (MX-80, FX-80, RX-80, LX-80, etc.), Diablo Advantage D25, Diablo C-150, Okidata Okimate 20, Hewlett Packard Laser Jet, Hewlett Packard Laser Jet Plus, and Qume LetterPro 20.

Even if you do not have one of the printers listed, your printer may work correctly if you choose one of the default printer settings. That is because many different printers share a number of the same escape codes. If you have a daisy-wheel printer, you might try setting your preferences for one of the daisy-wheel printers on the list, such as the Diablo 630 or Qume LetterPro 20. Several other manufacturers have tried to make their printers compatible with the escape codes they use. Likewise, among dot-matrix printers, Epson has set a de-facto standard by virtue of the widespread acceptance of its printers, including their adoption by IBM. Therefore, most of the new dot-matrix printers are to some extent "Epson compatible". These include such popular printers as the Star Micronics Gemini series, the Citizen 120D and the Panasonic 1090 and its kin. To the extent that these printers are truly Epson-compatible, you will obtain good results when you choose "Epson" as your printer type. You should be aware, however, that many printers that claim to be Epson compatible do not match every single special feature exactly. For one thing, there is not just one Epson standard. Earlier Epson models like the MX-80 do not have all of the same features and escape codes as the current models. Therefore, the Epson driver may not support every feature of your Epson-compatible printer correctly, particularly if your printer is a couple of years old.

If you can not get your printer to work with one of the settings on the list, you may have to install a custom printer driver. Select "Custom" from the list of printers, and then type in the name of the custom printer in the box marked "Custom Printer Name" that appears below the list of printers. The default name is "generic", which installs a printer driver that ignores all special features. If worse comes to worst, you should be able to get a plain, unformatted printout by using this driver. However, it may be possible for you to obtain a custom printer driver that fully supports the features of your printer.

There are several sources for such drivers. One of these is Commodore-Amiga itself, though its new releases of the Workbench. Just as version 1.1 added support for the HP Laserjet and Laserjet Plus, the upcoming version 1.2 adds support for the Apple Imagewriter II and Okidata ML92 printers. A second possible source is the manufacturer of your printer. Though there are few manufacturers that provide Amiga support for their printers (as Okidata did for the Okimate 20),

more of them would if the demand was greater. There are commercial sources for printer drivers as well. Digital Creations (the makers of Gizmoz) sell printer drivers for the Apple Imagewriter II color printer, as well as for 24-pin dot-matrix printers like the Toshiba P1351, P351, and P351-C printers, and the C. Itoh 24LQ. These printer drivers may be purchased with a printer cable for $65, or for $45 for the software alone. Software Supermarket makes available to Amiga dealers a program that they can use to create custom printer drivers. Dealers may buy the program for $150, and are granted an unlimited license to sell the drivers that they produce to their customers. The suggested price for these drivers is $35. In addition, Software Supermarket maintains a BBS that dealers may call during the evening to download drivers that have already been produced with the program, thus saving them the time and effort of constructing one themselves. Though Software Supermarket will not sell its printer-driver construction program to end users, they do maintain a listing of dealers that participate in their program. Therefore, if you call them during business hours at (716) 873-5321, they will be able to refer you to the dealer nearest you who can provide you with the printer driver that you need.

An excellent (and much cheaper) source for printer drivers is the public domain. You can find a number of printer driver files posted on commercial information services such as CompuServe, PeopleLink, Delphi, and GEnie, as well as on local BBS's. If you are equipped with a modem and a password, you can obtain these drivers for next to nothing. Another source is the Amicus public domain library, which is available from the Amicus group, or from Amazing Computing.

Amicus disk number 9 contains drivers for the Cannon PJ 1080A ink jet printer, the C. Itoh Prowriter, the NEC 8025A, the Star Gemini 10, the Okidata ML92, the Panasonic KX-P109x series, the Epson LQ800, and the Smith Corona D300. Additional drivers will probably appear on later volumes as well. Of course, if you are an adventurous (and fairly experienced) programmer, you could write your own printer driver. There are complete instructions and examples in Volume 2 of the ROM Kernel Manual. But you should be forewarned that it is not a trivial task. You must create several files in assembly language and/or C and link them together in order to produce a working driver. Nevertheless, customizing an existing driver to add special features should be within the grasp of most programmers.

After you've selected the custom printer driver from Preferences, don't forget to copy the actual printer driver file to the correct directory. In order for the PRT: device to use the driver, it must be located in the Devs/Printers directory of the Workbench disk you use to start the computer. Don't forget that the converse is true as well. If you only have one printer, you are free to delete all of the other drivers in the "Devs/printers" directory. This can save you some room on your Workbench disk.

Assuming that the volume name of the disk which contains the driver is DRIVER and the volume name of your Workbench disk is WORKBENCH, you could transfer the driver to your Workbench disk with the CLI command:

```
COPY DRIVER:MyDriver to WORKBENCH:Devs/Printers
```


Also, make sure that the name you entered in the "Custom Printer Name" box in Preferences is exactly the same as the name of the printer driver file.

In addition to selecting a printer type from Preferences, there are several other settings that you should attend to as well. The default connection for the printer is through the parallel port, so if you are using a serial printer hooked up to the modem port, be sure to click on the "Serial" icon in the upper left corner of the "Change Printers" screen. You may specify the length of the paper that you are using and whether it is continuous-feed fanfold paper, or single sheets. You may select line spacing of 6 or 8 lines per inch, and left and right margins settings. Also, you may choose the size of the type (Pica, Elite, or Condensed), and either draft or near-letter quality.

While all programs have access to these preference settings, they are free to adopt or ignore them. Some prefer to present their own menus allowing the user to select these options, and others use the expressed preference of the user as a default. Just remember that even though a program may have its own menu for margin settings and the like, it may also be affected by the Preference settings. Even some programs that print screen graphics may be affected by the margin settings. You will find that the size of the graphics printout can be affected by the margin settings. The moral of the story is, if you can't get a proper printout by adjusting the program settings, try changing the Preference settings as well.

The final printer settings that you can make from Preferences are on the "Graphic Select" screen. For most dot-matrix printers, the default settings are fine, and you need not make any changes. In some cases, though, you will need to make some adjustments. If you have a color printer, make sure that you select "Color" from the "Shade" box at the bottom-left of the screen. By default, screen graphics are printed horizontally, but for a larger image you can select to print them vertically as well. You may find in some cases that you get a better aspect ratio from vertical printing as well (pictures that look "scrunched" horizontally may print truer to the screen version vertically). Also, if you are printing a lot of images that consist of light colored patterns on a dark background, you may wish to switch to Negative printing, so that the pattern will be printed instead of the dark background. This can save a lot of time, as well as wear and tear on ribbons and print heads.

One final reminder about the Preference program. Be sure to exit the program with the Save option, in order to save your selections to disk. Once you have saved your preferences to one disk, you may transfer them to any other simply by copying the "Devs/System-Configuration" file from that disk to the "Devs" directory of the new disk.

Although using the PRT: device is the simplest way for most applications to communicate with the printer, we will briefly touch on the other methods available. In addition to PRT:, there are two other AmigaDOS devices that you can use to output data to a printer. These two are SER:, the serial device, and PAR:, the parallel device. To use these devices you follow the procedure outline at the beginning of the article. First, you must know how the printer is connected to the Amiga, so that you can choose either the parallel or serial port. Then, you open that device and send characters to it. You may do this directly from a program, or use one of the DOS commands like COPY or TYPE send the contents of a file to the device. The characters that you send to either port are not examined or translated in any way, so in order to invoke special printer features you must send the exact printer codes for the particular printer that you are using. Use of PAR: should be restricted to those situations where you can be sure of the kind of printer that will be used, such as for software that is to be used on one machine only. PAR: can also be very helpful in troubleshooting a printer problem. If you are unsure of whether the problem lies with your software printer driver or your hardware printer connection, you can send data directly to PAR:. Since this data is output directly to the printer without any translation, you can tell if your printer hardware is receiving it correctly.

The final method for accessing a printer with the Amiga is for programmers only. Experienced programmers who are comfortable working with hardware devices at the Operating System level can bypass the AmigaDOS printer devices, and address the hardware device "printer.device". This device gives the programmer complete control over the printer. It allows him to send Amiga escape sequences which will then be translated by the printer driver to the printer specific codes, or to send characters directly to the printer with no processing at all. In either case, the programmer need not know how the printer is connected to the Amiga.

The following BASIC program gives examples of sending printer escape codes through the PAR: device and through the PRT: device, both by opening the device directly, and by using the LPRINT command:

```
'*************************************************
'*   Printercode.bas  by Sheldon Leemon          *
'*                                               *
'*   This program demonstrates three ways to send *
'*   Escape codes to your printer from Amiga     *
'*   BASIC.  The first sends printer-specific    *
'*   codes to the PAR: or SER: device (depending *
'*   on how your printer is connected to the     *
'*   Amiga).  A program written using this method *
'*   will only work correctly with that specific *
'*   printer (in this case, my Toshiba P1351).   *
'*   Obviously, this method is not recommended   *
'*   unless there is no other choice.            *
'*                                               *
'*   The following sections send Amiga-specific  *
'*   codes to the PRT: device.  Programs using   *
'*   these methods will work with ANY printer    *
'*   that is correctly installed from Preferences. *
'*                                               *
'*************************************************


' Set up strings using printer-specific codes

A$ = CHR$(27)+"!"+"Enlarged Print is on"
B$ = CHR$(27)+CHR$(34)+"Enlarged Print is off"
C$ = CHR$(27)+"I"+"Underline is on"
D$ = CHR$(27)+"J"+"   Underline is off"

' Send them to PAR: device

OPEN "PAR:" FOR OUTPUT AS #1
PRINT #1, A$;"    ";B$;CHR$(13)
PRINT #1, C$;D$;CHR$(13)
CLOSE #1

' Set up strings using generic Amiga printer codes

A$ = CHR$(27)+"[6w"+"Enlarged Print is on"
B$ = CHR$(27)+"[5w"+"Enlarged Print is off"
C$ = CHR$(27)+"[4m"+"Underline is on"
D$ = CHR$(27)+"[24m"+"   Underline is off"

' Send them to PRT: device (also called LPT1:
' by Amiga BASIC).

OPEN "PRT:" FOR OUTPUT AS #1
PRINT #1, A$;"    ";B$
PRINT #1, C$;D$
CLOSE #1

' LPRINT them.  LPRINT sends its output to PRT:
' (actually, to BASIC LPT1:BIN device).   In order to
' make Amiga escape codes work, you MUST remember to
' send CHR$(0) or other non-printing character first.

LPRINT CHR$(0)+A$;"    ";B$
LPRINT CHR$(0)+C$;D$

END  'End of Printercode.bas
```

Although you may use any of the three methods shown for sending data to a printer, you should be aware that it may not always be possible to mix them in a single program. All three methods use the same printer port hardware. Therefore, it is not possible to OPEN the PAR: device when the PRT: is already open--to do so generates a "file already open" error. When dealing with PAR: or PRT:, it is at least possible to CLOSE one device before OPENing another. But using LPRINT OPENS the printer device and leaves it open even after the printing is complete. The only way to CLOSE the printer after an LPRINT statement is to issue a CLEAR, which closes all files, but also wipes out the value of all of your variables. This makes it impractical to try to open the PRT: or PAR: device in the same program which uses the LPRINT statement.

The final question that many Amiga owners have about using their printers with the Amiga is how to do an exact bit-by-bit copy of their graphics screen to the printer. This is commonly known as a screen dump. IBM and IBM-compatible personal computer have a special key combination that you can press to send the screen graphics to the printer at any time. The Amiga, however, does not have a similar feature. Instead, there is a built-in Operating System routine that automatically performs a screen dump. But you must have some program call to that routine. Some programs, like Notepad and Deluxe Paint, provide screen-printing facilities from a menu. But there are also a number of public domain screen-dump programs that allow you to send any graphics screen to your printer.

Amicus Disk #8 contains two of these programs, and one of them, Scrimper, was featured in a previous issue of Amazing Computing. And as this is being written, Discovery Software is readying its release of GRABBiT, a screen dump program that takes the same "hot key" approach as the IBM PC. This program sells for $29.95, and allows you to use a CTRL-ALT key combination to print the screen at any time.

TABLE 1: Amiga Printer Codes

Function	Escape Sequence*	Printers Supporting
Reset	c	Al Br D6 Cm Jx Ep Da Dc Ok Hp L+ Ql
Initialize	#1	Al Br D6 Cm Jx Ep Da Dc Ok Hp L+ Ql
Linefeed	D	Al Br D6 Cm Jx Ep Hp L+ Ql
Return + linefeed	E	Al Br D6 Cm Jx Ep Da Dc Ok Hp L+ Ql
Reverse linefeed	M	Al Br D6 Jx Da Ql
Normal character set	[0m	Al Br D6 Cm Jx Ep Da Ok Hp L+ Ql
Italics on	[3m	Jx Ep Ok Hp L+
Italics off	[23m	Jx Ep Ok Hp L+
Underline on	[4m	Al Br D6 Cm Jx Ep Da Ok Hp L+ Ql
Underline off	[24m	Al Br D6 Cm Jx Ep Da Ok Hp L+ Ql
Boldface on	[1m	Al Br D6 Cm Jx Ep Da Hp L+ Ql
Boldface off	[22m	Al Br D6 Cm Jx Ep Da Hp L+ Ql
Set Foreground Color	[nm	Br Jx Da Dc Ql
(n is a two-digit ASCII number from 30-39)		
Set Background Color	[nm	Dc Ql
(n is a two-digit ASCII number from 40-49)		
Normal pitch	[0w	Al Br D6 Cm Jx Ep Da Ok Hp L+ Ql
Elite on	[2w	Al Br D6 Cm Jx Ep Da Ok Hp L+ Ql
Elite off	[1w	Al Br D6 Cm Jx Ep Da Ok Hp L+ Ql
Condensed fine on	[4w	Al Br D6 Cm Jx Ep Da Ok Hp L+ Ql
Condensed off	[3w	Al Br D6 Cm Jx Ep Da Ok Hp L+ Ql
Enlarged on	[6w	Cm Jx Ep Ok
Enlarged off	[5w	Cm Jx Ep Ok
Shadow Print on	[6"z	Al Br D6 Cm Jx Ep Da Ql
Shadow Print off	[5"z	Al Br D6 Cm Jx Ep Da Ql
Doublestrike on	[4"z	Al Br D6 Cm Jx Ep Da Hp L+ Ql
Doublestrike off	[3"z	Al Br D6 Cm Jx Ep Da Hp L+ Ql
NLQ on	[2"z	Cm Jx Ep
NLQ off	[1"z	Cm Jx Ep
Superscript on	[2v	Al Br D6 Cm Jx Ep Da Dc Ok Hp L+ Ql
Superscript off	[1v	Al Br D6 Cm Jx Ep Da Dc Ok Hp L+ Ql
Subscript on	[4v	Al Br D6 Cm Jx Ep Da Dc Ok Hp L+ Ql
Subscript off	[3v	Al Br D6 Cm Jx Ep Da Dc Ok Hp L+ Ql
Normalize the line	[0v	Al Br D6 Cm Jx Ep Da Dc Ok Hp L+ Ql
Partial line up	L	Al Br D6 Cm Jx Ep Da Ok Hp L+ Ql
Partial line down	K	Al Br D6 Cm Jx Ep Da Ok Hp L+ Ql
US character set	(B	Cm Jx Ep Hp L+
French character set	(R	Cm Jx Ep
German character set	(K	Cm Jx Ep
UK character set	(A	Cm Jx Ep Hp L+
Danish I char. set	(E	Cm Jx Ep
Swedish character set	(H	Cm Jx Ep
Italian character set	(Y	Cm Jx Ep
Spanish character set	(Z	Cm Jx Ep
Japanese char. set	(J	Cm Jx Ep Hp L+
Norweigen char. set	(6	Cm Jx Ep
Danish II char. set	(C	Cm Jx Ep
Proportional on	[2p	Al Br D6 Cm Jx Ep Da Ok Hp L+ Ql
Proportional off	[1p	Al Br D6 Cm Jx Ep Da Ok Hp L+ Ql
Proportional clear	[0p	Al Br D6 Da Ok Hp L+ Ql
Set prop. offset	[n E	Br Da Ql
Auto left justify	[5 F	Al D6 Jx Ep Da
Auto right justify	[7 F	Al Jx Ep
Auto full justify	[6 F	Cm Jx Ep
Auto justify off	[0 F	Al D6 Cm Jx Ep Da Hp
Letter space (justify)	[3 F	Jx Ep
Word fill(auto center)	[1 F	D6 Jx Ep
1/8" line spacing	[0z	Al Br D6 Cm Jx Ep Da Ok Hp L+ Ql
1/6" line spacing	[1z	Al Br D6 Cm Jx Ep Da Ok Hp L+ Ql
Set form length n	[nt	Al Br D6 Cm Jx Ep Da Dc Ok Hp L+ Ql
Perf skip n (n>0)	[nq	Cm Jx Ep Da Ok Hp L+
Perf skip off	[0q	Cm Jx Ep Da Ok Hp L+
Left margin set	#9	Al Br D6 Da Dc Ql
Right margin set	#0	Al Br Da Dc Ql
Top margin set	#8	Al Br D6 Da
Bottom marg set	#2	Al Br D6 Da
T&B margins	[n1;n2r	Hp L+ Ql
L&R margin	[n1;n2s	Br D6 Cm Jx Ep Da Dc Hp L+
Clear margins	#3	Br D6 Cm Jx Ep Da Dc Hp L+ Ql
Set horizontal tab	H	Br D6 Da Dc
Set vertical tab	J	Br D6 Da
Clear horizontal tab	[0g	Br D6 Da Dc
Clear all horiz. tabs	[3g	Br D6 Cm Jx Ep Da Dc
Clear vertical tab	[1g	Br D6 Da Dc
Clear all vert. tabs	[4g	Br D6 Cm Jx Ep
Clear all h & v tabs	#4	Br D6 Cm Jx Ep Da Dc
Set default tabs	#5	Br D6 Cm Jx Ep Da Dc
Extended commands	[n"x	

Notes

The abbreviations used for the printers are:

```
Al = Alphacom Alphapro 101
Br = Brother HR-15XL
D6 = Diablo 630
Cm = Commodore CBM MPS 1000
Jx = Epson JX-80
Ep = Epson  X-80 series
Da = Diablo Advantage D25
Dc = Diablo C-150
Ok = Okidata Okimate 20
Hp = Hewlett Packard Laser Jet
L+ = Hewlett Packard Laser Jet Plus
Ql = Qume LetterPro 20
```

* All escape codes must be preceded by the ESC character (ASCII 27 or CHR$(27) in BASIC). In addition, when sending escape codes to the PRT: device via the Amiga BASIC LPRINT statement, you must precede the ESC character with a non-printing character, such as CHR$(0). For example, to send the code for "Underline on", you would send character 27 to the PRT: device, followed by the character string "[4m". In BASIC you could accomplish this with the statement: LPRINT CHR$(0)+CHR$(27)+"[4m";

n In the table above, a lower case n stands for a decimal number, expressed as a set of ASCII digits, for example "12". For example, to set the form length to 50 lines, you would send ASCII character 27, followed by the string "50t". In BASIC this would read: LPRINT CHR$(0)+CHR$(27)+"50t";

•AC•

Forth!

"If you need a special tool for a special job
...make it ***yourself***."

by Jon Bryan

Tools

I've heard it said that the best tool is the one that "comes easily to hand." Expert craftsmen have a habit of making their own tools for just that reason. What better way to make a tool "come easily to hand" than to build it yourself? Woodworker, machinist, or programmer, the principle is the same. If you need a special tool for a special job, often the best (and maybe the only) way to get it is to make it yourself.

For decades programmers have created and refined high-level languages to make them "come easily to hand." One approach is to generalize and try to design a language to be all things to all people. The Forth philosophy is to provide a number of simple tools which can be used to build more complex ones.

As I promised a couple of installments back, I'm working on a three-dimensional simulation of a bouncing ball. At first I thought that I would probably use Blitter Objects (BOBs) for the ball images but I've since learned a few more things. I had thought that hardware Sprites were restricted to three colors plus transparent but upon receiving my long-awaited copy of the Hardware Reference Manual I found that I was wrong.

The Amiga actually has two kinds of hardware Sprite. A "simple" sprite is restricted to three colors, but there is another type called an "attached" sprite which can have fifteen colors (plus transparent). After some study I decided that I needed to build some tools for the creation of both simple and attached sprites. I would like to show you these tools and explain a few of the things I learned about sprites while creating them.

Sprites, to quote the manual, are "hardware objects that are created and moved independently of the playfield display and independently of each other." The mouse cursor is a sprite, by the way. There are eight sprite channels, allowing eight simple sprites to be created, or four attached sprites. A sprite is always 16 pixels wide but may be any number of scan lines tall. Sprites may also be reused within a display field, allowing more than eight sprites to be displayed on the screen under special circumstances. The size of a sprite is based on a pixel that corresponds to the low-resolution non-interlaced (320x200) display mode, which is also the resolution for movement of the sprite.

A sprite is stored in the Amiga's memory as pairs of 16-bit words. The first pair of words are control words which contain information on the position at which the sprite is to be displayed and how tall it is. Following the two control words are the pairs of words which constitute the sprite image. There is one pair for each scan line. Following the image are two more control words which are used when the sprite is reused within the same display.

If the numbers 0 through 3 represent the available colors, a sprite might be represented like this:

```
0000122332210000
0001223333221000
0012233333222100
0001223333221000
0000122332210000
```

This is not the way the sprite is stored in memory, however. Each line of the sprite image must be translated into two 16-bit words. The first line of the example sprite would be written into memory as the following binary values:

```
0000100110010000,0000011111100000
```

Each bit of the first word provides the least-significant bit of the color selector number, while the second word provides the most-significant bit. In the following representation each pair of digits represents the color selector for one pixel.

```
  0 0 0 0 1 0 0 1 1 0 0 1 0 0 0 0  <--First word
  / / / / / / / / / / / / / / / /
  0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0  <--Second word
```

A simple sprite's two bit planes restrict the number of colors to four, one of which is transparent, and each sprite has a fixed set of color registers. Sprite 0 uses registers 17-19, sprite 1 gets its colors from registers 21-23, etc. Registers 16, 20, 24 and 28 aren't used by simple sprites because those values are always transparent.

If more colors are desired it is necessary to use an "attached" sprite, which is created by superimposing two simple sprites. This effectively doubles the number of bit planes and allows

the use of sixteen colors. The odd-numbered sprite of the pair must have the "attach" bit set in the second control word, which causes the hardware to use the bits in both sprites to select the color for each pixel.

The Hardware manual gives this example of an attached sprite image.

```
0000154444510000
0001564444651000
0015676446765100
0001564444651000
0000154444510000
```

and here are the corresponding data words for the first line and the sprite they're assigned to

```
      0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0  <--1st word of even sprite
     / / / / / / / / / / / / / / / /
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  <--second word of even sprite
   / / / / / / / / / / / / / / / /
  0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0  <--first word of odd sprite
 / / / / / / / / / / / / / / / /
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  <--second word of odd sprite
```

The odd-numbered sprite provides the most significant bits of the color selector. Attached sprites use color registers 17-31 (16 being transparent). The restrictions which apply to attached sprites are that only four sprites may be used and that the sprites must be consecutive pairs. Sprite number 1 may be attached to sprite 0, sprite 3 to 2, and so on.

The ROM Kernel calls which manipulate hardware sprites are GetSprite, FreeSprite, ChangeSprite and MoveSprite. Once the appropriate data structures are created and initialized these calls handle all the details of assigning and moving hardware sprites.

That's it for this installment. I direct your study to the Hardware Reference Manual for more details including sprite priority and reuse. For the rest I'll let the code speak for itself.

```
\ Hardware Sprite Tools   08-11-86:jrb

DECIMAL
256 CONSTANT ScanBufSize
CREATE ScanBuf ScanBufSize ALLOT

\ The definition for "SpriteLine" is primitive.  It
\ will parse 16 characters composing one scan line
\ of the sprite and leave the starting and ending
\ address of the resulting string.  It's not smart
\ enough to handle comments on the same line as the
\ sprite data unless the comment precedes the data.

: SpriteLine   ( -- addr1\addr2 )
   ScanBuf ScanBufSize INFILE @ READ.TEXT   1- (
trim delimiter )
   ScanBuf + DUP 16 - ;

\ "?SpritePixel" attempts to convert a character
\  into a legal value for a sprite pixel.

: ?SpritePixel   ( character\base -- value )
   DIGIT NOT  ERROR" Illegal Sprite Color" ;

\ Each character in the sprite definition will
\ represent two bits of data, one for each bit
\ plane.

\ "OR_SpritePlanes" will break out those bits and
\ rotate them into place.  Remember that the data
\ for a simple sprite is stored as pairs of 16-bit
\ words,  with the first word representing the
\ "least-significant" plane.

: OR_SpritePlanes   ( number\address -- )
   SWAP  2 /MOD     ( separate the two bits)
   SWAP  16 SCALE   ( slide low-order bit up a word)
   OR               ( put them back together)
   OVER @  2*       ( move stored value to the left)
   OR  SWAP !    ( and OR the new bits into place.)
;

\ "Sprite" expects a height parameter which
\ represents the number of
\ scan lines in the simple sprite.

: Sprite   ( height -- )
   DUP 4* 8+
   CREATE  HERE
     LOCALS|  image  size  height  |
     size ALLOT  image size ERASE
                 \ allocate and clear the space
                   4 ADDR.OF image +!
                 \ step over SPRxPOS and SPRxCTL
                   height 0
     DO  SpriteLine
       DO  IC@ 4 ?SpritePixel  image
OR_SpritePlanes
       LOOP  4 ADDR.OF image +!
     LOOP ;

\ The following image is courtesy of the Hardware
\ Reference Manual:

5 Sprite ShipImage
   0000122332210000
   0001223333221000
   0012223333222100
   0001223333221000
   0000122332210000

\ Now we define an occurence of a SimpleSprite
\ structure named "Ship" and initialize it.

struct SimpleSprite Ship
  ShipImage Ship +ssPosCtlData !
  5 Ship +ssHeight W!
  0 Ship +ssX W!
  0 Ship +ssY W!
  0 Ship +ssNum W!
structend

\ The sprite may be displayed by executing the
\ GetSprite and ChangeSprite system calls, as
\ follows:
\
\   Ship 1 GetSprite (get the one you ask for)
\   ViewAddress +vViewPort @  Ship  ShipImage
\   ChangeSprite
\
\ and using the following definition to move the
\ sprite around.

: MoveShip   ( x\y -- )
   LOCALS|  y  x  |
   ViewAddress +vViewPort @  Ship  x y  MoveSprite
;
```

```
\ "Attached" sprites are a little bit more complex.
\ First a data structure:

structure AttachedSprite
   simpleSprite STRUCT:  +asEvenSprite
   simpleSprite STRUCT:  +asOddSprite
structure.end

\ An attached sprite has four bit planes which it
\ builds by "attaching" the two bit planes of an odd-

\ numbered sprite to those of an even-numbered
\ sprite.  The two sprites used are defined in the
\ same way that a simple sprite is, except that the

\ odd-num sprite has its "attach" bit set (bit 7
\ of its second control word).  The other
\ restriction  is that consecutive sprites must be
\ used.  In  other words, sprite 1 can be attached
\ to sprite 0, sprite 3 to sprite 2, etc.

\ Since there are four planes, breaking out the
\ bits is more complex. The odd-numbered sprite
\ contains the most-significant bits.

: OR_AttachedPlanes(char\even sprite\odd sprite-- )
   LOCALS|  odd  even  |
  DUP 4/  odd OR_SpritePlanes \ shift the two MSB s
   3 AND  even OR_SpritePlanes ;
            \ mask the two lowest bits

: Attached   ( height -- )
   DUP 4*               \ Number of bytes in image
   8+                   \ Plus control words
   DUP 2*               \ Total size
   CREATE  HERE
   LOCALS|  image  size  offset  height  |
     offset 2+ W,
            \ lay down offset to "attached" image
     2 ADDR.OF image +!
              \ increment pointer into array
     size ALLOT  image size ERASE     \ make room
     128 image offset + !
          \ set "attach" bit in odd-numbered sprite
     4 ADDR.OF image +!
          \ increment over SPRxPOS and SPRxCTL
     height 0
     DO  SpriteLine
       DO  IC@ 16 ?SpritePixel
            \ Base 16 allows characters 0-F
         image  image offset +  OR_AttachedPlanes
       LOOP   4 ADDR.OF image +!
     LOOP ;

\ Now a couple of words to extract the addresses
\ of the images.

: +EvenImage    ( addr1 -- addr2 )
                  \ leave address of even sprite
image
   2+ ;

: +OddImage    ( addr1 -- addr2 )
                 \ ditto for the odd-numbered image
   DUP W@ + ;

\ The values for the following image were calculated
\ using an equation gleaned from "Graphics and Image
```



```
\ Processing" by Theo Pavlidis.  It is arguable that
\ you could do as well freehand.  In fact, I _have_
\ touched up the image a bit.

15 Attached Ball_Image
   000000BBBB000000
   0000CDEEEEDC0000
   008CDEEFFEEDC800
   00ACDEEFFEEDCA00
   07ABCDEEEEDCBA70
   079ABCCDDCCBA970
   4689ABBBBBBA9864
   457899AAAA998754
   4456778888776544
   0344566666654430
   0334444444444330
   0033344444433300
   0033333333333300
   0000333333330000
   0000003333000000

struct AttachedSprite Ball
  Ball_Image +EvenImage Ball +asEvenSprite
+ssPosctldata !
  Ball_Image +OddImage  Ball +asOddSprite
+ssPosctldata !
  15 Ball +asEvenSprite +ssHeight W!
  0 Ball +asEvenSprite +ssX W!
  0 Ball +asEvenSprite +ssY W!
  0 Ball +asEvenSprite +ssNum W!
  15 Ball +asOddSprite +ssHeight W!
```

```
   0 Ball +asOddSprite +ssX W!
   0 Ball +asOddSprite +ssY W!
   0 Ball +asOddSprite +ssNum W!
structend

\ Executing this definition will set up the colors
\ for the ball.  It will also change the color of
\ your mouse cursor.

: 17-31.Greys    ( -- )
                  \ Only for registers 17 through 31
    ViewAddress +vViewPort @
    16 1 DO
      DUP  i 16 +  i i i  SetRGB4
    LOOP  DROP ;

\ The procedure for getting an attached sprite on
\ the screen is more complex.  Remember, you have
\ to use consecutive sprites, and the odd-numbered
\ sprite has to be the "attached" sprite.
\
\  Ball +asEvenSprite  2  GetSprite .
\  Ball +asOddSprite   3  GetSprite .
\  ViewAddress +vViewPort @  Ball +asEvenSprite
\  Ball_Image +EvenImage
\  ChangeSprite
\  ViewAddress +vViewPort @  Ball +asOddSprite
\  Ball_Image +OddImage
\  ChangeSprite

\ One detail which I haven't yet been able to find
\ documented is that when MoveSprite is used to move
\ the even-numbered sprite in the attached pair, it
\ also moves the odd-numbered sprite.  That's why
\ the following definition works.

: MoveBall    ( x\y -- )
    LOCALS|  y  x  |
    ViewAddress +vViewPort @  Ball +asEvenSprite  x y

       MoveSprite ;

\ Please keep in mind that the code presented here
\ represents tools. It obviously won't do much by
\ itself.  I'm working on incorporating all this
\ into a demo, but it will be a little while yet.
\ If you want to have some fun, write a little
\ routine to  move the sprite  around the screen
\ continuously, then grab the Workbench drag bar
\ and move the screen up and down.  Try it when the
\ sprite isn't moving, too.
```

•AC•

Amazing Reviews...

Easyl
from Anakin Research, Inc.

Tired of drawing with the mouse
...then this pad may do the trick

by Keith Conforti
AC Art Director

Recently, Amazing Computing™ ran a test on Easyl™, a newly developed graphics tablet. Easyl™ is created by Anakin Research Inc.™. The kit includes a pressure-sensitive pad, a bus connector, and an Easyl™ program disk. The pad and bus are also compatible with Aegis Images™ and DeluxePaint™.

The pressure-sensitive pad for Easyl™ is connected to the bus connector box which is attached to the Amiga bus on the side of your machine. Easyl™ responds to any pointed instrument, but it is recommended that the user works only with a dull no. 2 pencil. The tablet also has a sixteen color palette, clear command, erase command, in-fill command, and save command, all activated by finger touch keys. The drawing surface is 8 1/2 by 11 inches, standard to most sheets of paper.

Easyl™'s Software

When we first received Easyl™, we tried running it with its own program. Loading the program was fairly simple and quick, but its capabilities were rather limited. The response from the pad was somewhat sensitive, but also seemingly sluggish. I found the fingertip functions to be very temperamental and repetitive. At times I could not change colors without entering the command four or five times before succeeding, especially with the higher numbered (9-16) colors on the palette. Also, the in-fill function would sometimes erase previous, unsaved in-filled shapes, and the save would not always register new lines.

The instructions call for a "light touch" on the pad keys, but that did not always remedy the situation. As a result, using the Easyl™ program provided with the pad was, at times, very frustrating. However, I did persist long enough to create the school-house on this month's cover.

Easyl™ is provided with this program and its source code. The source code is a gateway into creating a drawing package that fits your needs. That is, if you are capable of doing your own programing.

DeluxePaint™

Not very satisfied with the Easyl™ program as it stood, I decided to use the DeluxePaint™ program to see if the results would be any better. Loading Easyl™ with DeluxePaint proved to be no easy task. The copy protection of DeluxePaint™ made it almost impossible to install the drivers on the disk (who wants to fool with their master disk). We never could manage to get that particular program to run with Easyl™.

Ageis Images™

However, when I tested Easyl™ with the Aegis Images™ program, the turnout was quite different. First of all, I loaded it using both drives and the process became quick, simple, and efficient. Once the program was loaded, Easyl™ took on all of the commands of Aegis Images™ (a considerable improvement over Easyl's limited commands). The response from the pad was more receptive and faster, and the images were truer to what I had drawn. But probably the biggest advantage of using Aegis Images™ over Easyl™ was the mouse.

On the Easyl™ program disk, the mouse does not work on anything screen-related, only for edit and color functions, etc. But with Aegis™ the mouse can work along with the pad on the screen. This saves the user hours of tedious time when making straight lines, circles, squares, and many other drafting related shapes, and the graphics are much more exact than when using a ruler on the pad. I enjoyed using the program with Aegis™ more than any of the others because it made the work a lot less frustrating and a lot more fun.

The Verdict

The final verdict on Easyl™ is a fair one. In itself, the software program provided is not that great, you could do just as well with any other graphics program and a mouse. However, the source code is provided, allowing you the option of creating your own software package .

Using Easyl™ with another program, probably Aegis Images, the result is a pleasing combination of your own talent and the latest in computer graphics.

Easyl™
by Anakin Research
$499.00

•AC•

Christmas Fantasia

Christmas Fantasia is a very special collection of over 40 popular and traditional Christmas pieces that plays continuously while displaying beautiful Christmas scenes. The music has all the features of our SYMPHONY LIBRARY. Christmas Fantasia plays 'real' sampled instruments for music realism and can also be connected to a MIDI synthesizer for additional flexibility. Playing time over 1 hour.

Guaranteed to get even Scrooge into the spirit of Christmas.

Christmas Fantasia #CF 921 . . . . . **$29.95**

Symphony Library

Over 30 Hours of Music!

Each volume of 100 songs of the SYMPHONY LIBRARY turns the AMIGA into an incredible musical JUKEBOX for the music lover and gives the musician the ability to modify and enhance the music using either the AMIGA's powerful sound reproducing capabilities or those of external MIDI synthesizers. All songs are in IFF format to insure compatibility with existing and future software. The sampled "real" instruments are also in IFF format thereby allowing the user to add to the instruments library using the user's own sampled instruments or public domain instruments.

- Over 100 songs in each volume.
- Full 4 voice arrangements.
- Over 3 hours of music in each volume.
- Stereo, Mono, and MIDI output simultaneously.
- User selectable "sampled" IFF instruments.
- Volume control slider.
- Tempo control slider.
- Transpose up or down 24 half steps. (2 octaves up and down).
- Select single or multiple MIDI channels.
- User friendly mouse driven menu interface.
- A potpourri of music in each volume.
- Compatible with MIDI synthesizers.
- Jukebox feature allows you to define sequence of songs.
- 8 Volumes to choose from.

SYMPHONY LIBRARY Each volume of 100 songs. (Specify Vol. 1 through Vol. 8) **$39.95**

List of all 800 songs. **$3.95**

MIDI Connection

The MIDI CONNECTION is a hardware interface to allow you to connect the AMIGA to any MIDI synthesizer, drum machine, sampler and more.

- MIDI IN
- MIDI OUT
- MIDI THRU
- Two 6' MIDI Cables
- Music Software Included
- 30 page manual
- Connects in seconds.
- Complete list of all 800 songs of the SYMPHONY LIBRARY
- Connects to the AMIGA serial port.
- Guaranteed compatible with all MIDI software (MUSIC STUDIO, SOUNDSCAPE etc.)

MIDI CONNECTION #MC 923 . . . . . **$39.95**

6 MIDI OUT's

Deluxe MIDI Connection

2 MIDI IN's

We developed the **DELUXE MIDI CONNECTION** for those with multiple synthesizers. It has all the features of our standard MIDI CONNECTION, with the additional feature of giving 6 MIDI OUT's and 2 MIDI IN's.

DELUXE MIDI CONNECTION #DM 924 . . . . . **$79.95**

We accept CASH, CHECK, COD, VISA and MASTER CARD orders.
Shipping and handling US and Canada $3.00
Shipping and handling outside the US and Canada $5.00
COD charge $2.00
Illinois residents add 6¼% sales tax.

VISA MasterCard

Dealer Inquiries Invited

MUSIC STUDIO is a trademark of Activision.
Sound Scape is a trademark of Mimemics.
AMIGA is a trademark of Commodore-AMIGA

Speech Systems
38W255 DEERPATH ROAD
BATAVIA, ILLINOIS 60510
(312) 879-6811

AmigaNotes

*MIDI and the Amiga...an **excellent** combination.*

By Richard Rae
Music Editor
CIS# 72177,3516

Myths and MIDIocrity

Last month we took a very brief look at MIDI and some of the things it can do for you in your electronic music system. Just to make sure you understand that MIDI is not all sweetness and light, this column presents the opposing viewpoint: problems. While we're poking around the back alleys of MIDI I'll also try to explain away some of the persistent myths.

(Before you begin reading this, be advised: these topics are controversial and sometimes lead to very heated discussion. One of the statements I'll be making, for example, is in direct disagreement with a column which recently ran in another magazine. These are the facts as I see them; others may or may not agree. So if casually mentioning something you read here gets you into an argument, don't say you weren't warned. <Grin>)

The Stuck Note Syndrome

This seems to be the most talked-about problem with MIDI, probably because it's the most obnoxious. The Stuck Note Syndrome has been described as "When you stop playing but the synthesizer doesn't"; a fairly accurate description at that.

Cast back to last month's column when we discussed **Channel Messages,** and you'll recall there were two messages called **Note On** and **Note Off.** The first is used to initiate a note on a synthesizer, the second to terminate a note. Bet you can guess what happens if, for some reason, a synthesizer misses a Note Off message!

There are several ways this can happen. The obvious one is for an entire Note Off message to somehow disappear, but this is both unlikely and unnecessary. What happens if just ONE bit of a Note Off command is changed?

If the MSB of the status byte is changed, the command becomes a byte of data and the entire string will, one would hope, be ignored. If one of the three command bits is toggled, our Note Off command is no longer a Note Off command and will be misinterpreted. If one of the channel bits is altered the message will be ignored altogether (unless the synthesizer has been programmed to read all channels). Change one of the data bits in the note number and the wrong note will turn off... no help, or worse. Finally, change the MSB of the data byte and it becomes a command. What command?

Who knows! So, we have at least sixteen critical bits in a Note Off command, any one of which can totally negate the command.

I can imagine howls of outrage from a few readers. "That's a ridiculous system. Haven't these people ever heard of parity? They could at least DETECT the error!"

True, there is no parity bit supported by MIDI. What good would it do? Remember that the MIDI cable carries data one way only, so a synthesizer couldn't ask the computer to retransmit a message if it detected an error; the best it could do is ignore it, which is little improvement over what we have now. Error correcting codes are a possibility, but they would impose a performance penalty on the CPUs in the synthesizers, as well as strain the already limited MIDI bandwidth. There is no obvious way out with the current MIDI specification. Fortunately, Stuck Note Syndrome is acceptably rare in a well planned system.

But what causes dropped bits and stuck notes? There are several possibilities including extremely noisy electrical conditions, bad cables, instrument buffer overflow, and so on. By far the most common cause, however, is MIDI Data Distortion. Which segues nicely into topic two:

MIDI Data Distortion

To discuss this problem we need to once again examine MIDI hardware.

Many MIDI synthesizers are equipped with three ports: MIDI In, Out, and Through. The functions of In and Out are obvious; the Through port presents a buffered copy of the data entering the In port. This makes daisy-chaining of instruments very straightforward.

Each MIDI In port receive its data via an opto-isolator to provide isolation between various pieces of equipment. One of the limitations of opto-isolators is that there is a significant time delay involved in turning a phototransistor on. This means that a square wave into an opto- isolator will be a poorly formed pulse wave coming out. Take this pulse wave through another opto-isolator, and the pulses become worse yet. Continue this process over and over, and you will eventually reach a point where the pulses simply disappear.

This is exactly the situation we have when we daisy-chain synthesizers. Each MIDI In port drives an opto-isolator, resulting in a slightly degraded version of the MIDI data being presented to the Through port, which feeds the next MIDI In port, which drives an opto-isolator...

And the data pulses don't have to go away completely; only a little distortion is necessary before a UART or ACIA returns gibberish. The upshot of which is that if you try to daisy-chain a dozen MIDI devices you may be treated to all sorts of unpleasant surprises including, yes, Stuck Note Syndrome.

The cure for Data Distortion is simple and inexpensive: don't daisy-chain beyond a reasonable limit, say, four devices or so. If you want more synthesizers on line, you need to invest in another piece of equipment: the MIDI **Through Box.**

A Through Box is an outboard accessory which, in its basic form, has one In port and multiple Through ports. Rather than daisy-chaining a series of devices, a Through Box allows you to set up a **star network**, where the Through Box becomes the central distribution point. One cable carrying data from your computer or sequencer is buffered and split into multiple outputs, which are Through ports in MIDI vernacular. This means that the signal at each output has been passed through only one opto-isolator.

Through boxes are fairly inexpensive and eliminate Data Distortion even in very large systems. As an example, consider a Through Box with four Through ports and three synthesizers daisy-chained from each port. With this arrangement you could drive TWELVE devices with no more distortion than that caused by daisy-chaining four devices directly from the computer.

MIDI Delay

This problem is one of MIDI data being delayed before it reaches some of the instruments, resulting in an audible time lag. This can be anything from barely discernible to totally debilitating.

MIDI is a serial system which operates at a rate of 31,250 bits per second. Each data byte is bracketed by one start bit and one stop bit, resulting in an actual data rate of 3,125 bytes per second or 320 microseconds per byte.

Most MIDI messages are two or three bytes long. For example, a **Note On** or **Note Off** message is comprised of three bytes: the **status byte** indicating the command and two **data bytes** indicating the note number and velocity.

A good rule of thumb for estimates, then, is that each note event requires one millisecond to transmit. (In reality there are methods which can shorten this time significantly).

This is the absolute maximum speed at which MIDI data can be transmitted. This rate is quite adequate for real-time human performance, but what happens if you are multi-tracking with a computer and several events should occur simultaneously?

Consider a hypothetical case: what if you had a lead line, two four note chord strata, a bass part, and a percussion section. If we assume that two percussion instruments might be struck simultaneously -- high hat and base drum, for example -- then we could expect up to 12 events simultaneously. Due to MIDI bandwidth, this could result in up to 12 milliseconds of delay in some voices, which is bordering on perceptible.

In addition to Note On messages, continuous controls such as pitchbend and aftertouch can eat up horrendous chunks of time. Still more bandwidth is required for system real-time messages (responsible for timing and sync when using a sequencer). And we haven't even considered the fact that for every Note On we have to find time to squeeze in a Note Off. Add them all together and you could be listening to a serious problem.

There are several ways to eliminate MIDI Delay, including filtering and half-speed mastering. The most general and useful technique, however, is to use a star network.

In a true star network, each instrument has its own dedicated MIDI cable. Since there is no daisy-chaining involved, each cable can carry ONLY the data for the connected instrument. Thus the serial data travels in parallel from the computer to the synthesizers.

Only a few systems are currently capable of supporting a true star network. The scheme requires the computer to separate the data for each instrument and send it to its individual output, plus the hardware to provide those individual outputs. As MIDI users become more sophisticated and demanding I expect we will see more of these systems appearing.

Instrument Delay

The symptoms of Instrument Delay are identical to MIDI Delay... an audible time lag between when a note should sound and when it actually does. It is caused by time delays within the instrument itself. To be honest this is not a MIDI problem, but due to the similarity of symptoms Instrument Delay is often confused with MIDI Delay. Typically both contribute to any system delay problems.

Today's keyboards are thousands of times more complex than the synthesizers of just ten years ago. As computers have progressed from the Altair with its binary front panel switches to the Amiga, so have synthesizers advanced from hand-patched systems like the early Moogs to today's DX7s. Synthesizers in fact owe their current sophistication to the microprocessor industry, as they have been riding on its coattails for the last several years.

For the most part today's keyboards are made more powerful simply by having the onboard processors DO more. But there is a price to pay. Between scanning the keyboard and front panel, controlling parameters, generating envelopes, and perhaps even generating the waveforms themselves, there is precious little time left for handling MIDI messages. Some instruments tend to get a bit overloaded, and as a result it might take a few milliseconds for the CPU to get around to processing a MIDI message.

One possible cure for Instrument delay is the star network. If the instrument can only look at the MIDI data stream every so often, it must make best possible use of the available time. If data for all the synthesizers in a system are travelling down one MIDI cable, then the CPU in each instrument will be taking quite a bit of time just saying "Oh, this message isn't for me". By using a star network and sending only the data for that instrument we can eliminate this wasted time.

Another approach is **filtering**. With this technique we take the data on a single MIDI cable and filter out the nonessential information before it reaches the instrument. Some of the recently announced Through Boxes include an onboard CPU and support this sort of function.

MIDI Choke

This is a problem which was once labeled as "The Dreaded MIDI Choke"; I liked the phrase so much I've used it ever since. It is an extreme case of bandwidth limitation either in the MIDI data stream, the computer, or the instruments themselves. It is characterized by stuttering, very long delays, strange instrument reactions, and even lockup of the instruments or the computer itself.

Since the CPU in an instrument cannot react instantly to MIDI data, it is usually stored in a buffer until it can be handled; the same is true on the computer's end. What happens if data comes in so quickly that the buffer overflows? At the very least data will be lost. This is, in fact, another possible cause of Stuck Note Syndrome. In less robust systems the CPU could get confused trying to interpret the data, resulting in who knows what result.

On the computer's end, what if data is being sent faster than the MIDI bandwidth allows? Once the transmit buffer fills, where does the extra data go? Should the computer just throw it away and keep going, or wait for the buffer to empty? Tough decisions for the author of the sequencer program.

MIDI Choke is an extreme and unique condition; the only sure cure is patience and experimentation. You may be able to solve the problem with a star network, since this allows data to exit the computer more quickly than does a single MIDI cable. Filtering out unwanted data when recording or playing back could help if you can afford to lose some data. Other more drastic methods are available as needed.

What, Me Worry?

So there you have some of the most obnoxious MIDI problems. Should you, wide-eyed with fear and loathing, take your newly acquired MIDI interface at arm's length and flush it down the toilet? No. With a small system you may NEVER experience any of these problems. Larger systems simply require careful planning to avoid pitfalls.

Realize, too, that few people have enough money to go out and buy a dozen synthesizers and drum machines and hook them all up at once. If you are like most hobbyist musicians (myself included), you will be adding new equipment slowly, one piece at a time. Problems will tend to crop up slowly and gently after a new acquisition; this gives you the opportunity to puzzle out exactly what happened and how to fix the problem.

The information in this article is presented as a hedge, preventative maintenance if you will. If a problem strikes and something presented here hints at the proper solution, then it has served its purpose.

So much for enumerating problems. Now let's take a quick look at a couple of MIDI myths.

MIDI MYTH #1: "MIDI Delay is caused by the time lag between In and Through ports in each device."

Many people believe that the MIDI data at the In port is processed by the instrument's CPU before being passed to the Through port, resulting in a delay of several milliseconds. Cascade several synthesizers using Through ports, and presto: instant MIDI Delay.

In reality the CPU has nothing to do with the MIDI Through port. The data at the In port is buffered by hardware, typically one or two TTL gates, before being presented to the Through port. At worst, this results in a delay of a few nanoseconds and does not contribute to the problem.

MIDI MYTH #2: "A Through Box will cure MIDI Delay problems."

The belief that a Through Box eliminates MIDI Delay has two sources. The first is based on MIDI Myth #1: If the problem is caused by cumulative port delays, then eliminating the daisy-chain will cure the problem. The second reason for the Through Box myth is based on a misunderstanding of star networks. Understand this well: A Through Box Doth Not A True Star Network Make. Remember that the idea behind the star network was to increase the rate at which data could get to the synthesizers. With a Through Box all the data still has to go down one MIDI cable to get TO the distribution point; the bottleneck remains.

A true star network has to be driven by a high speed input. This generally means the computer itself. Put a bunch of UARTS on the buss and drive them independently from the sequencer program and you have a TRUE star network. Plug a MIDI interface with two MIDI Out ports into your Amiga's serial port and you do NOT, since the data still has to get to the interface via the 31.25 Kbaud serial port.

And that, my friends, concludes this summary of the negative aspects of MIDI. All in all MIDI fares pretty well: the capabilities and flexibility make the occasional headaches well worthwhile, especially considering how the same jobs were done just a few scant years ago. I hope you'll have an opportunity to work with MIDI and the Amiga; I expect it will be an excellent combination.

A closing comment...

Several people have mentioned that they are desperate for music hardware and software, and have asked me to include projected release dates and announcements of availability each month. I think it is safer to decline this request, for a couple of reasons.

First, we all know how tenuous and unreliable estimated shipping dates are. I'd almost consider it a disservice to give you shipping dates which are later pushed back.

Second, even though Don Hicks and his staff are doing an incredible job of getting Amazing out the door, the fact is that I have to get my articles in one month before the cover date. Add to this the composition time at the publisher's offices, the printing delay, and distribution. Many projected release dates will have passed by the time the magazine is in your hands. You'll already KNOW before I can TELL you!

I think this is one of the areas where the networks fit in. For something as volatile as this subject, services like CompuServe (which I strongly recommend) and Plink (which is John Foust's favorite) are the answer. Come join us online for the latest information!

This refusal notwithstanding, I really **do** want to know what you would like to see here. Remember, this is your column, your magazine. If you don't tell me what you want, the best I can do is shoot in the dark.

Nybbles,
Rick

The Amazing AmigaBASIC Tutorial

Part Six ... Screens and Windows

By Kelly Kauffman

For your Amiga to put anything on your monitor in MBasic, there must be a screen with a window open on it to put it in. You may or may not be familiar with screens and/or windows, but this tutorial will devote itself to explaining them to you as they pertain to MBasic.

SCREENS

The screen command will create a new screen in which future windows may exist. A screen defines how many colors, and what resolution a window will have.

The syntax of the command is:

SCREEN [screen #],[width],[height],[depth],[mode]

Screen # is a number from 1 to 4 which, when windows are later opened, will tell the Amiga which screen to put the window on.

Width will be a number between 1 and 620. This will specify, in pixels, how wide your screen will be. The two "normal" widths are 320 for low-resolution (maximum 32 colors) and 640 for high-resolution (maximum 16 colors). The Workbench is an example of maximum screen width. If you specify a number other than 320 or 640, your screen will not fill the monitor horizontally. If, for instance, you specify a width of 300 in low-res, you will have an inch or so on the right hand side of your monitor that will be a "void." You will not be able to drag windows into the unused 20 pixels, nor will you be able to put anything in there. One interesting thing is that when you move the mouse into that area, it disappears!

Height is a number between 1 and 400. If the number is above 200, then you must later specify an interlaced window (more on this later.) The normal screen height for a non-interlaced window is 200. Again, the Workbench is a good example of this, it is a 640 x 200 window. If you specify a height that is not 200 or 400, you can get some strange results. First off, when the screen is opened up with a height of, say 100, the bottom half of the screen is garbled garbage. There's jumbled graphics on the lower half which will disappear as soon as you click on the screen's title bar. When you do click on the title bar, the screen JUMPS to the lower half of the screen, thus fixing the display. Screens of non-normal heights (normal=200 and 400), will always appear from the bottom up. In other words, you can slide the screen down off the bottom, but you can't slide it off the top.

Depth is a number which specifies the number of "bit-planes" the screen will have. This is a fancy term which basically determines the maximum number of colors that a screen will be able to display as shown in the following chart:

Depth	Max. # of Colors
1	2
2	4
3	8
4	16
5	32

In high-resolution, the maximum depth number you can use is 4, since the systems hardware is limited to 16 colors in hi-res. In lo-res, you can specify a depth of 5, allowing you to display 32 colors.

You may be wondering why the other depths are provided. Well, the "deeper" the screen, the more the Amiga has to work on the screen to keep it drawn on your monitor. Thus, it slows the machine down...which is very noticeable in MBasic. For example, a math intensive program will run fastest on a 320 x 200 screen with a screen depth of 1. Not as glossy, but faster. The same program on a 640 x 400 screen with a depth of 4 will be NOTICEABLY slower. The Workbench has a screen depth of 2.

Mode is a code for MBasic that tells it exactly what kind of resolution you want. For example, you may have a 320 x 200 lo-res screen which fills the monitor, or you can have a 320 x 200 hi-res screen which will only take half the width of your monitor. This lets you tell the Amiga what resolution you want according to the chart below.

Mode	Result	
1	Low Resolution,	NOT interlaced
2	High Resolution,	NOT interlaced
3	Low Resolution,	interlaced
4	High Resolution,	interlaced

Interlace is that funny mode that makes your display "flicker" a bit. But in a sense, you gain virtually another screen since it is like having the area of two screens in one. You MUST specify interlace if you have a number larger than 200 for your height. There are some tricks you can use to minimize the side-effects of the interlace mode. Careful color planning will virtually eliminate the flickering effect. (Use the PALETTE command to change the colors. For best results, use darker colors with nearly the same brightness. The Palette command will be covered in depth in a later issue.)

You also MUST specify a mode of either a 2 or a 4, for hi-res, if your screen width is greater than 320.

Some sample **SCREEN** commands:

```
SCREEN 1,320,200,5,1
```
A lo-res 320 x 200 screen with 32 colors

```
SCREEN 1,640,200,4,2A
```
hi-res 320 x 200 screen with 16 colors

```
SCREEN 1,320,400,3,3
```
A lo-res 320 x 400 screen with 8 colors interlaced

```
SCREEN 2,640,400,1,4
```
A hi-res 640 x 400 screen with 2 colors interlaced

```
SCREEN 3,280,325,3,4
```
A hi-res 280 x 325 screen with 8 colors interlaced

To close a screen and free up the memory that it takes, then use the command:

SCREEN CLOSE [#]

The # is the screen id# that you want closed. (Note: Do not type the brackets around the number.) Any windows that were open on the screen you closed will automatically be closed.

WINDOWS

Windows allow you to print or draw things on your SCREEN. You cannot draw things on a screen directly from MBasic...they MUST be in a window. This is the syntax for the window command:

WINDOW [window id],["title"],[size],[type],[screen-id]

Window id is just a number to keep the windows separate in case you have more than 1 window and want different information in each window. You will then be able to specify which window you want to modify by specifying it's window id #. Window #1 is the Output window which appears when you first enter MBasic. Any number above 1 is a window that you have created (or will create).

The title must be contained in quotes ("") but is optional. If you do not specify a name, the title bar of the window will be blank.

The size of the window is the coordinates of the upper-left hand corner and the lower-right hand corner of the window. For example, if you wanted a window that appeared 10 pixels over and 15 pixels down and went over to 290 and down to 185 then the syntax for the size argument would be: (10,15)-(290,185)

If you don't specify the size of the screen, then it defaults to the maximum size that the SCREEN can handle.

The Type determines what "options" you want for your window. This is the chart of options for windows:

ValueOption

1 A sizing gadget will be in the window so that the size of the window can be changed by the user.

2 The window can be moved by clicking and dragging on the title bar.

4A Front/Back and Back/Front gadget will appear in the window so it can be moved in front of or behind other windows.

8A Close gadget will appear in the upper-left corner of the window so the user can closethe window.

16 If another window is moved on top of this one, and then moved off of the window, it's contents will re-appear. This option eats up memory, because it sets aside enough memory for the window twice. One visible, the other, somewhere in memory. This slows down the windows since when writing or drawing into the window, it has to be drawn on actually two windows to have a reference as to what to put back into the visible window after a window has been moved off of it.

In order to have more than 1 option per window, you simply select which options you want in a window, add their "Values" up, and plug it into the "TYPE" argument of the window command.

Screen id tells the Amiga which screen to open this window on. This is the same number that you specified in the Screen command as the *Screen #.* If you specify a screen id of -1, the window will be opened on the Workbench screen.

Some sample Window opening commands:

```
WINDOW 1,"System Data", (10,10)-(270,180),15,2
```

- This opens a window called "System Data" that can be sized, closed, dragged and move foreground/background on screen #2.

```
WINDOW 3,,,,
```

- This opens a full window with no title, with no options on the workbench screen.

To close a window, use the command:

WINDOW CLOSE [#]

The # is the *window id* # to close. (Note: Do not type the brackets.) This will free up any memory used by the window.

The **WINDOW OUTPUT [#]** command will force a window to the foreground and make it the current output window. (Note: Do not type the brackets.) This means that any PRINT, CIRCLE, etc. commands will be done in this window.

Note that with both the Screen and Window commands, that each argument MUST be separated with a comma.

That'll do it for Screens and Windows, I hope this helped you understand them better. Good luck!

•AC•

Amazing Reviews...

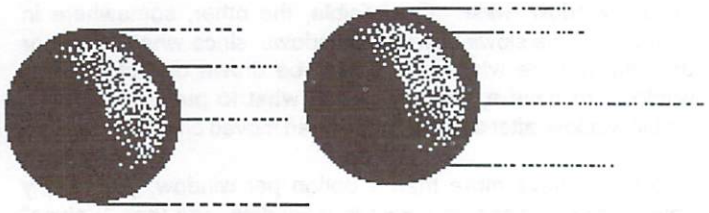
Marble Madness

Coin-Op Comes to the Amiga with a Superb Release from Electronic Arts

Reviewed By
Stephen R. Pietrowicz

The program that many Amiga owners have been waiting for is finally here: Marble Madness.

Marble Madness was released by Atari as a coin-op game several years ago. The object of the game is to manuever your marble through a series of 3-D raceways to the goal line. The mazes you have to guide your marble through have ramps, funnels to send your marble from one part of the maze to another, and a large variety of hazards that you'll have to get your marble past. There are six levels to navigate, and the quicker you make it through one level the better. The time left from the previous level is added to the time on your next level.



Hazards include "Steelies", black marbles that try and knock your marble off the edge of the maze. If you're able to knock off a "Steelie" instead, you'll be awarded bonus points. Another type of opponent are little worms called "Marble Munchers" that stun your marble, try and eat it, and lick their "lips" if they succeed. Moving pools of acid, vacuums, and other obstacles are scattered throughout each level.

Marble Madness can be played as either a one or two player game. You control the marble by using a joystick, mouse, or a trackball. (Both types of trackball are supported). You can mix and match the types of controls that you use in the two player version of the game. If both players use a mouse, make sure that there is lots of room! By pressing on the left mouse button, (or the button on a joystick or trackball), it gives the ball an extra burst of speed. I've found that doing this while using a mouse really helps to control the marble.

Moving the marble can be frustrating at times, depending on which type of control device you use. The joystick is great for moving along a straight path, but it's a bit difficult to use around sharp corners. (I think it is a matter of getting used to it more than anything else). When using the mouse, keep it near the center of the table. Otherwise, you lose control as soon as the mouse goes off the edge of the table.

Unfortunately, there are some things I don't really like about the game. The mouse MUST be attached to the front port in order to select different controls, and to start the game. If two players both use joysticks, they have to reattach the mouse to the front port, click on the "GO!" to start the game, and then quickly switch back to using the joystick before the game starts. It would have been easier to just be able to hit the space bar to start the game, and not have to worry about switching back and forth.

Once one level of the maze has been completed, the next level has to be loaded from disk. Depending on the level, this can take up to 20 seconds to do. It would have been nice to at least keep the music running while waiting for the new level to load.

I've even had the game "guru" on me once. In a particularly ungraceful move, my marble was hit by a "Steelie" and it bounced from one section down to another section, and then to another. Up came the guru message. I've tried to recreate the problem, but had no success.

Despite these things, I've enjoying playing Marble Madness. It was certainly aptly named. I've gone crazy trying to get past obstacles on the way to the end of the race!

•AC•

Electronic Arts
1820 Gateway Drive
San Mateo, CA 94404

Copy protected disk $49.95
512K; Joystick, trackball, or mouse required
One or two players

Using Fonts from AmigaBASIC

...fonts & libraries and how to use them from AmigaBASIC

By Tim Jones
People Link AMSOFT 1
SYSOP The Window BBS, (617)-868-1430

Doing things the hard way has always appealed to me. I was perusing my ROM Kernal manuals and saw that the font calls should be a challenging way to spend a couple of days working in AmigaBASIC. After reading the sections in the RKM and in the Sybex 'Amiga Programmer's Handbook' concerning fonts, I started to work.

As usual, the folks at Commodore had already thought about this and included a simple example on the Extras disk called 'Library'. I took a look at what they offered and thought 'That shows me the system fonts, now what about the fonts that are included on the disk?'

I got to poking around in the text functions section of the Amiga Programmer's Handbook and found the call to OpenDiskFont(). My next move was to scan the entries in the diskfont.bmap file [see Amicus disk #8 in the .BMAP Files section or on Fish Disk #27 in the ABDemos directory] using my BmapReader program to see if the function was included.

It was there that I also found the function AvailFonts(), which had been referred to in the OpenDiskFont() function description. AvailFonts() then led me to AllocMem() and FreeMem().

For library work, BmapReader is probably the most useful thing that I have ever done. Turning to it and the .bmap files, I located the functions that I was looking for and then turned back to the Amiga Programmer's Handbook for full descriptions.

After collecting all of the information concerning fonts, I then looked at the C programming example WhichFont.c in the ROM Kernal manual and decided that, if they could do it in C, I could do it in BASIC. WhichFont.Bas is the resulting program.

Program Description:

The library functions called by WhichFont.BAS are as follows:

```
AllocMem()          -> from the exec library
FreeMem()           -> from the exec library
OpenDiskFont()      -> from the diskfont library
AvailFonts()        -> from the diskfont library
OpenFont()          -> from the graphics library
CloseFont()         -> from the graphics library
AskSoftStyle()      -> from the graphics library
SetSoftStyle()      -> from the graphics library
```

Therefore, we must open the exec, diskfont and graphics libraries. You must have the exec.bmap, diskfont.bmap and graphics.bmap files in your current directory or in the SYS:Libs directory for this program to work.

Variables used by the program, along with short descriptions are:

```
BufPtr&     -> Ptr to the allocated mem for the font list
BufSize&    -> size (in bytes) of the allocated buffer
MEMFPUBLIC& -> a flag indicating that the memory allocated
               shall be fully public and relocatable
MEMFCLEAR&  -> flag the alloc mem should be clear to zeros.
FontList&   -> record call to AvailFonts. Zero if all well.
AFFDISK&    -> Tell AvailFonts to look at the disk fonts
AFFMEMORY&  -> Telling AvailFonts to look at resident fonts
NumFonts%   -> # fonts located by the AvailFonts call
NewPtr&     -> ptr to base the font name starting location
StrAdd&     -> ptr to address that the font name is held at
Num%        -> counter for the FOR-NEXT loops
PointSize%()-> array holding the size of the matching font
PointPtr&   -> ptr to the size info in textattr structure
AvailName$()-> string array holding the actual font names
Ft$         -> string for user entry of font for viewing
Pt%         -> the point size of the font to be viewed
Type%       -> the af_Type (memory (1) or disk (2))
Rp&         -> pointer to my rastPort for WINDOW 2
enable%     -> tells SetSoftStyle which styles are available
               for the font called.  Acts as a bit-mask.
pFont&      -> pointer to the currently active font
fontName$   -> string in the Font Sub-Program for passing
               the name of the font to be opened.
fontName0$  -> fontName$ with a CHR$(0) termination
height%     -> the point size of the font being called
style%      -> the style of the font.  I use zero here and
               call to SetStyle to change the style.
prefs%      -> this tells the OpenFont() call the preferred
               style(s) for the font you are calling.
textAttr&(0)-> ptr to the adrs of the fontName0$ variable
textAttr&(1)-> the height, style & preferences for the font
mask%       -> passes the style to the SetStyle Sub-Program
```

AllocateMem

The first step in setting up our fonts list structure is to allocate some memory for storage of the structures as they are returned from the AvailFonts() call. We do this by calling AllocMem() and passing the values for the amount of memory we require (bytesize) and the type of memory (requirements). These can be described as follows:

(Please note that BASIC ignores the underline character, so I leave it out.)

bytesize
Number of bytes required. Rounded up to the next multiple of eight bytes if not on an even eight byte block boundary.

requirements
MEMF_PUBLIC, MEMF_CHIP, MEMF_FAST or MEMFCLEAR (These are ORed together and passed)

MEMF_PUBLIC
value = 0 (no bits set) This indicates that the memory allocated must not be mapped, swapped or otherwise unaddressable. Since we would eventually want our structure list to be used by other tasks, we must use PUBLIC memory.

MEMF_CHIP
value = 1 (bit 0 set) This indicates that the memory must be in the first 512K bytes of system memory (chip memory). Allows access of the memory by the custom chips, DMA, etc.

MEMF_FAST
value = 2 (bit 1 set) This indicates that the memory should be in the memory located above the first 512K bytes of system memory. This memory may only be accessed by the 680XX MPU and not the custom chips.

MEMF_CLEAR
value = 65536 (bit 16 set) This indicates that the memory should be cleared, set to all zeros, upon allocation.

My AllocMem() call looks like this:

```
BufPtr& = AllocMem&(BufSize&,(MEMFPUBLIC& OR MEMFCLEAR&))
  IF BufPtr& = 0 THEN
    PRINT "Couldn t find";BufSize&;"Bytes of contiguous free
  RAM."
    FreeMemExit
  END IF
```

If the call could not locate 512 bytes of contiguous RAM, then a zero would have been returned and we would have exited the program. Otherwise, BufPtr& will contain the address of the memory that was allocated for our use.

Notice that the variables are passed as LONG (&). This is required by the system to keep things agreeable.

GetAvailFonts

The next step in the process is to find out what fonts are available on the disk and in memory. We do this with the AvailFonts() call. The syntax of the call is:

error = AvailFonts(bufptr, numbytes, where)

Bufptr The address of the memory the list structures should be created in.

numbytes The number of bytes available

where Where the call should look (memory or disk or both) where can be either AFF_MEMORY (1) or AFF_DISK (2).

My call to AvailFonts() looks like this:

```
FontError& = AvailFonts&(BufPtr&,BufSize&,AFFDISK&
  OR AFFMEMORY&)
  IF FontError& <> 0 THEN
    PRINT "Couldn t find Fonts.  Cleaning up..."
    FreeMemExit
  END IF
```

If the call to AvailFonts() doesn't work, FontError& will contain an error number other than zero.

GetFontNameAddresses, GetTheType, GetThePointSizes and BuildNames

We will assume that the AvailFonts() call worked and we now have our fonts list structures in the memory that we allocated. The first structure, AvailFontsHeader, starts at location BufPtr& and contains two bytes listing the number of fonts located according to our call to AvailFonts().

The rest of the entries are the AvailFonts structures. Each entry in the list is ten bytes long. The byte definitions are listed below. If we are running on a standard WorkBench disk, there should have been 16 fonts returned (14 on disk and 2 in memory).

The next three routines in the program, GetFontNameAddresses, GetThePointSizes and BuildNames, get their information from these structures. Before looking at them, I set the variable NumFonts% equal to the two byte value returned in the AvailFontsHeader structure.

We now look forward at the structures that contain the actual font information. The data is held in the following format:

bytes 1 - 2
The type of font (disk (2) or memory (1)).
This value is assigned to the variable Type%.

bytes 3 - 6
The address of the font name in memory. We look there to get the name for use in our DisplayNames routine.

bytes 7 - 8
The Ysize of the font (Point size). This tells us how many pixels high the font is.

byte 9
The style bits for the font (describes which styles are applicable to this font).

byte 10
The preferences (the way the font SHOULD be displayed).

DisplayNames, GetDecision and DisplayFont

Once all of this data has been sorted and placed into arrays, we will display it in a human readable form. This is performed by the DisplayNames routine. The information displayed lists the name of the font, the point size and the type (memory (1) or disk (2)).

After the names, point sizes and types are displayed , the user is asked to enter the name, point size and Type of the font that he/she wishes to see. The name can be just the name (i.e. topaz, sapphire, etc.) or the name with the .font extension. the format of the entry would be:

Enter the name of the font, its point size & Type from the chart above:

```
Font Name, Point size and Type (separated by
  commas): topaz,11,2

Type END,0,0 to exit
```

This would then open a new window and display the selected font in all fifteen possible style combinations. Once the font has been displayed, the user is prompted to click the MOUSE button to continue. He/she is then returned to the list of fonts and prompted for the next font. To exit the program, the user enters 'END,0,0' as the font name and size.

SUB-Programs

If the user exits, we call the sub-program FreeMemExit. This frees up the memory that we allocated in the beginning, closes any open fonts using a call to CloseFonts() and closes the libraries. The main call in this sub-routine is the call to FreeMem(). Since FreeMem() doesn't return a value, we didn't declare it in the DECLARE FUNCTION section. The items passed to the FreeMem() call are MemPtr and MemSize. MemPtr is our variable BufPtr& and MemSize is our variable BufSize&.

My call to FreeMem() looks like this:

```
SUB FreeMemExit STATIC
  SHARED BufPtr&,BufSize&,pFont&
  IF BufPtr& <> 0 THEN
    CALL FreeMem&(BufPtr&,BufSize&)
  END IF
```

Also in this sub-routine is a call to CloseFont, which is needed to insure that all system resources (memory) are returned before we exit. We base our call to CloseFont on the value held by the variable pFont&. If we find that pFont& isn't zero, then a font must still be open. The call to CloseFont() looks like this:

```
IF pFont& <> 0 THEN CALL CloseFont(pFont&)
```

AMIGA® HAS MULTI-TASKING, DISCOVERY SOFTWARE USES IT!

FROM NOW ON YOU CAN PRINT OR SAVE ANY SCREEN, FROM ANY PROGRAM, ANY TIME!

GRABBiT takes WYSIWYG* to the limit. With GRABBiT you capture exactly what you see on your screen in an instant, regardless of what other programs you're running. GRABBiT works with all AMIGA video modes, including "**Hold-and-Modify**". It even lets you capture images from animated programs, like the bouncing ball in Boing! What's more, GRABBiT **runs completely in the background** – transparent to your other software. GRABBiT is **always** ready for you to use, even while you're in the middle of another program. As if that's not enough, GRABBiT requires only about 10K of your precious RAM to operate, and it supports dozens of printers. It's not a game, it's not a toy. GRABBiT is truly a productivity power tool for your AMIGA!

We believe powerful software should be easy to use. GRABBiT **is one of the EASIEST programs you'll ever use!** Every GRABBiT operation is triggered by one of the "HotKeys", a set of easy-to-remember key sequences that only take minutes to learn. Each HotKey is generated simply by holding down the "**Control**" and "**Alt**" keys and pressing one of the designated letter keys. What could be easier?

You won't grow old waiting for GRABBiT to finish printing, either. When we say multi-tasking, we mean it. GRABBiT has a **unique TPM (Task Priority Monitor) module** which makes sure your other software can still run even while GRABBiT is printing. The TPM module constantly tracks GRABBiT's printing priority, making sure it is neither too high nor too low, **but always just right!** GRABBiT adds a new dimension to the AMIGA's multi-tasking capability.

GRABBiT supports **dozens of different printers** because it uses the standard Amiga device drivers. Any printer you can choose in "Preferences" is **automatically** supported by GRABBiT. You'll get the most from color printers too, because GRABBiT supports full-color printing. In fact, we have seen amazing color printouts produced by GRABBiT on the Oki-Mate 20, a color printer costing less than $200.00.

Of course, GRABBiT's abilities are not limited **merely** to printing. GRABBiT is equally adept at **saving screen images to disk** – yes, even HAM screens! All GRABBiT disk files are saved in the popular **IFF format**, the emerging graphics standard for AMIGA. You can capture any screen to disk for slide-show presentations or later enhancement with any popular AMIGA graphics editor like AEGIS Images or Deluxe Paint. We even include a specially modified PD utility called "SEE" which allows you to view IFF image files quickly and easily. GRABBiT's disk operations are **lightning fast** because GRABBiT is written in a hybrid of highly optimized C and 68000 Assembler.

Once you start using GRABBiT **you'll want it on every disk**. You can easily install GRABBiT in your system startup-sequence, so it will **always be there when you need it**. With all its features this would be a great package at any price. But we think you'll agree with us that GRABBiT's most outstanding feature is **VALUE!** You get all the power of this sizzling new software for an unbelievably low

$29.95

+ $5 Shipping & Handling

VISA MasterCard

With bonus program, ANYTIME

GRABBiT®

*WHAT YOU SEE IS WHAT YOU GET

Amiga is a registered trademark of Commodore Business Machines
Images is a registered trademark of AEGIS Development Corp.
Deluxe Paint is a registered trademark of Electronic Arts.
Oki-Mate 20 is a registered trademark of Okidata of America

DISCOVERY SOFTWARE INTERNATIONAL

262 South 15th Street
Suite 400
Philadelphia, PA 19102

(215) 546-1533

"I have to give Marauder a hearty thumbs up and would recommend this product to friends."
AMAZING COMPUTING

"an indespensible tool for anyone with an Amiga"
INFO Magazine

You can buy the **DISK ANALYZER** we use for ½ price with Marauder II purchase

MARAUDER II®

Second Generation Excellence!

The Best Gets Even Better.

MARAUDER II is the **most powerful** copier for Amiga, no other copier even comes close. Copies most advanced copy-protection schemes ON ANY DRIVE with NO ADJUSTMENT REQUIRED, even SkyFox, Arcticfox, and the new Deluxe Series from Electronic Arts. **Automatic copying** is now **fully supported**, but Parameter-Entry override is still provided. MARAUDER II even includes new disk analysis utilities that will allow you to explore disks like never before.

MARAUDER II can also make **completely unprotected** copies of many of your favorite programs. No key disk is required after these programs are unprotected by MARAUDER II. If you plan to get a hard disk for your Amiga, then you'll really appreciate this feature!

Don't wait any longer. Join the thousands of Amiga owners who already rely on MARAUDER and get MARAUDER II today. Still not copy-protected, still only

$39.95 VISA MasterCard

$5 SHIPPING AND HANDLING

DISCOVERY SOFTWARE INTERNATIONAL

262 South 15th Street, Suite 400
Philadelphia, PA 19102 U.S.A.
(215) 546-1533

For my Font sub-program, I am using a slightly doctored version of the sub-program Font that is found in the Library program on your Extras disk. We send the same information as in the Library demo, but we make our open call based on the type of font (memory or disk). If the type (our variable Type%) is equal to two, then we call OpenDiskFont(). If Type% is equal to one, then we use OpenFont() and only look at the System resident fonts. We call Font as follows:

Font "fontname.font",height,style,preferences

Example: `Font "topaz.font",8,0,0`

To call Font from your program, you must initialize the following:

Any string or string variable containing the font name
The point size (as an integer)
The style and preferences (usually zeros on opening the font)
The type (memory(1) or disk(2))
The pointer to the rastPort (Rp& in this example or WINDOW(8))
The font pointer returned by OpenFont()/OpenDiskFont() (pFont&).

Once these variables are accessable by the sub-program, you may import it directly as presented into your program.

First, we check to see if a font is already open. If so, we close it before continuing to the OpenDiskFont call. The multiplication in the routine is simply to make the total value of textAttr&(1) equal to the total value of the last four bytes of the AvailFonts structure as discussed above.

Once we have the font open, we will display it using all fifteen style combinations. Please note that not all fonts were designed to operate in all styles. Most fonts were designed to be NORMAL, BOLD or UNDERLINED and therefore, they may not look right in the italicized or extended modes. To change the style of the font for display, we use a call to the SetFont sub-program. This sub-program requires that the variable enable% be set according to the return from the call to AskSoftStyle().

Programming Comments

To recap, opening fonts in your program requires routines that allocate memory (AllocMem()), check for available fonts (AvailFonts()), open and close the fonts (OpenDiskFont(), OpenFont() and CloseFont()), check and change the style of the font (AskSoftStyle() and SetSoftStyle()) and then return memory upon exiting. These are the AllocateMem: and GetAvailFonts: routines and the Font, SetStyle and FreeMemExit sub-programs in this example.

There is a set of new fonts in the public domain; I suggest to leave them alone. I have tried with all 28 fonts and the result has always been a visit from the GURU. Apparently, there is something unusual in the TextFont structures that define these new fonts.

I hope that I have shed some light on fonts and libraries and how to use them from BASIC.

•AC•

Roomers

John Foust guest hosts for the Amigo who is somewhere underground

By John Foust

Surprise! I am NOT the Amigo. I am substituting for the Amigo this month. I am not afraid to put my name above these words.

Commodore 64 emulator

I heard a rumor that someone had made a Commodore 64 emulator for the Amiga. I heard it from a former Commodore employee, who confidently whispered in my ear at spring COMDEX, and said I could call later, at the hotel, and hear the rest of the story. They also claimed an Apple][and Mac emulator were in the works, too.

Many weeks later, I heard this rumor again, on a computer network. Someone had talked to their dealer who had talked to a good friend of a person who was actually developing the C-64 emulator.

I smugly suggested that this person call their dealer, and have them call their friend of a friend, and have him call me, just to confirm this rumor. I thought this would stop the rumor in its tracks. I supplied my phone number in the message. The next morning, the developer of the C-64 emulator called me.

It is being made by a company called Software Kingdom, at 122 Prospect Hill Road, East Windsor, Connecticut, zip 06088, phone (203) 627-8180. It is not multitasking. It takes over the machine. At last rumor-posting, it was to ship the second week of September, at $149.95, and later rise to $199, after an introductory period.

The software emulates the 6510 in the C-64, and it includes a modified copy of the C-64 ROM, licensed from Commodore. It has a hardware pass-through add-on for the parallel port, like so many other Amiga peripherals. It gives a connection to the C-64 serial bus, so you can use your 1541 or 1571 disk drive or 1526 printer to your Amiga. It also provides some form of 'dongle' hardware copy protection.

Amiga Developer Conference

In the last few days, the Amiga Developer Conference was cancelled. Somewhere between 200 and 300 developers sent $300 registration fees to Commodore-Amiga, and a week before the conference, they cancelled it. The rumors of the cancellation started on Friday, September 5, and continued over the weekend. The rumors came with reasons.

One. A Macintosh developer conference was scheduled for the same week, and many developers prepare products for both machines. As one person said, "Let them move their conference!"

Two. Commodore management disapproved of the agenda for the conference, which included advance information on upcoming Commodore-Amiga products. They didn't want any leaks, so they asked Commodore-Amiga to cancel the conference. Many sources indicated a lot of internal political pressure.

This stands to reason. Even if every developer had signed a stack of non-disclosure agreements, word would hit the networks during the conference itself, and detailed information would soon follow in the press. After all, developers can use the phone, and many carry a portable computer.

The reasoning behind this reasoning is less clear. Somewhere in the Rulebook of Corporate Thinking, less information is better than more information - even if you are talking about the people who have invested their time and fortunes on your products, those people betting on the success of the Amiga, the developers.

Perhaps they imagined competitors would get a six-week jump on comparable products, or that the press would once again trample Commodore for announcing products before they are ready. I am sure Atari has better spies than this, and that the press will always enjoy trampling Commodore, the same way they like to trample Atari.

The developers were asked to stay at the Double Tree Inn in Monterey, California, the planned site of the conference. To confirm or disprove the rumor, I called the Double Tree on Monday night, a week before the conference. The night clerk had not heard about the cancellation.

COMDEX conflicts

On Tuesday, I talked to Caryn Havis-Mical, the conference organizer at Commodore-Amiga in Los Gatos. She confirmed that the conference is re-scheduled for November 5 through 7, the week before winter COMDEX in Las Vegas.

It will be held in Monterey, and everyone's registration fees and hotel reservations will transfer to the new date, and refunds are available. She was upbeat. She said they'd be able to produce a much better conference, if they only had more time to prepare.

Publisher's note: "Roomers" is an attempt to ferret out truths from half truths. We hope to make these remarks as accurate as possible. If some item is either missing or incorrect, please contact us with the corrected information.

The remarks in this column are those of Mr. Foust and not neccesarily those of PiM Publications, Inc.

They felt the extra six weeks would give them more time to prepare presentations for the conference. They could give a better conference in November than September.

However, the new date poses some conflicts. Many developers use the last week before a show to polish their product. COMDEX is the week after the newly scheduled conference, in Las Vegas. This could cramp the style of many developers, who need time to prepare for the show.

At that time, her office was addressing Mailgrams, and trying to call every registered developer, to alert them of the rescheduling. A cynical registrant said, "The letters should reach them about the time they get off the plane in Monterey." Another had heard of the conference cancellation, but had never received a phone call, or even an acknowledgement that his $300 arrived safely.

I suspect a few registrants will arrive in Monterey on the planned date. When the hotel tells them of the cancellation, they will rent a car, drive to Los Gatos, pound on the door at Commodore-Amiga, and ask for a special, private conference, in a strong tone of voice. They will get a visitor badge, and spend most of Wednesday morning in the lobby...

Commodore profits

Many newspaper readers were surprised when they opened their papers. 'Commodore' and 'profit' used in the same sentence? Commodore reported a profit of $1.2 million this quarter, which amounts to a few cents a share. With this announcement, the CBM stock price soon increased by more than a dollar.

New Amiga products

What was to be shown at the developer conference? I spoke with an anonymous Commodore-Amiga employee. Their words seemed to say that one-third of the planned presentation wouldn't be running in time for the conference, and they didn't want to make a bad impression.

They were worried about making a bad impression, and then they cancelled the conference? Developers would certainly understand if the hardware or software was buggy. I am sure many would be satisfied with cardboard mock-ups, and simple but accurate descriptions of the future product.

Several close sources confirmed that both the low-end and high-end Amiga products would not be functioning in time for the conference. What were these products?

Baby Amiga

By best accounts, a 'baby Amiga' was to be shown. This Amiga is dubbed the Amiga 500, or the 'P-52', according to one source. It is a consumer version of the Amiga. 'Consumer' means 'cheaper', I think, since this unit has an expected price of about $500. Some software developers have played with prototypes.

Rumors are it is an integral unit. The keyboard is part of the system unit. It has a disk drive on the side edge, like the Atari ST. It has the Kickstart version 1.2 in ROM, and is otherwise 100% percent software compatible.

It might not have provisions for an external monitor, only TV-set-quality video output. It has 256 or 512 K of RAM, with some provisions forinternal RAM expansion. It does not have a bus interface connector, so other kinds of expansion aren't possible. It doesn't have an internal fan, and it uses an external power supply, which prevents many FCC clearance problems. Unless Commodore has a large inventory of the present Amiga chip set, I'd guess it will have the new graphics chips.

These reductions should be able to bring the price into the $500 range. I wouldn't expect to see this machine until well into next year.

The 'P-52' name sounds like a ringer code-name, to me. In the past, Commodore has given employees false code-names for new products, to point out information leaks. Sometimes, the information spreads so fast, they can't pin down the leaks...

The Amiga 2500

Others claim the Amiga 2500 would be shown. Rumors give this machine various configurations, such as a built-in Sidecar daughter board, both 3 1/2 and 5 1/4 inch disk drives, and a 68010 processor. The rumors converge on a similar machine shown to select distributors at the spring COMDEX.

Ranger-class machines

'Ranger' was the code-name for future Amiga products. These upgraded Amigas have 68010 or 68020 processors, and the

previously announced graphics chips, with higher resolution, and greater bit-shuffling capabilities, along with the ability to address 2 megabytes of CHIP memory, as compared to the 512K CHIP memory limit in the present Amiga.

Some sources give the Ranger-class machines an internal hard disk. Developers would undoubtably applaud a hard disk, since so many are working from floppy disks. I suspect November conference attendees will see something with a DMA hard disk.

Sidecar rumors

By best accounts, the Sidecar has failed FCC clearance at least once. This has delayed shipment.

One rumor said the German engineers knew the Sidecar would fail FCC clearance, but submitted it anyway.

Another said the Sidecar is currently for sale in England, Europe and Australia. Another said it is on sale in Canada, but others say it is currently under test by the CSA, the equivalent of our FCC.

CBM was told by their own staff that the product could not pass FCC inspection, and they brought in an outside FCC expert, whose tests also failed, but they sent it in anyway. There were FCC specialists on the CBM West Chester staff, but they were laid off.

Don't expect to buy a Sidecar before January, according to smart folks. Dealers might have demo units before then. Several demo Sidecars are making the rounds at user group meetings in the United States and Canada.

Another rumor circulated, contrary to this. The Sidecar actually passed FCC clearance, despite rumors to the contrary. Commodore is keeping mum, to maximize the surprise when the Sidecar ships ahead of schedule.

This is hard to believe. Taking the slapstick of Commodore marketing as a given, I don't think they could keep this one under their collective hats.

However, this crazy rumor correlates to another Sidecar rumor. Someone has a friend who works for Commodore's advertising agency. They saw a four-color separation for an advertisement offering an Amiga and a Sidecar for $1395, in a "back-to-school" promotion.

AmigaDOS 1.2

So many people are excited about version 1.2 of the operating system. I'm not sure why. People were exchanging beta copies like wildfire - beta 2, beta 4, beta 5, beta 6, beta 7, and even rumors of a beta 9 flowed on the networks during the summer months.

Some developers got 5-disk sets of beta 2 and beta 4 from Commodore-Amiga, others got beta 6, but not 2 or 4. All of these were quickly copied at West coast user group meetings, and spread across the country.

In fact, Commodore sent a memo to dealers, warning them not to distribute the 1.2 betas as operating system upgrades. Jumping on this, Compuserve's Online Today, an electronic magazine, reported that Commodore was recalling the Amiga operating system. Sheesh.

Version 1.2 gives many more features, such as faster disk access. This is less in the realm of rumor, since so many people have copies of 1.2.

The developer disk set contained full online documentation of the changes and bug fixes present in each release. A future article in Amazing Computing will detail the new commands, so keep your shirt on, don't complain to me if your 1.2 betas crash a lot, and wait for the real release.

1.2 rumors

The disks were accompanied by many 1.2-related rumors.

One, that MetaComco is going to sell a BCPL compiler. Presumably, some developers could use it to fix AmigaDOS. This language isn't very popular in the United States, but more so in England. This is backed up by their own advertisements in other Amiga magazines.

Two, that Commodore-Amiga has re-hired MetaComco programmers to fix and improve AmigaDOS. Another source denies this rumor, and claims the original rumors misunderstood the completion of the MetaComco contract, and the contract work being done by former Los Gatos programmers.

Amiga Day November 15

10:00AM to 6:00PM

Featuring:

Minetics
Micro Systems Software
Electronic Arts
Amiga Controled Lasar Show

Guest Speaker R.J. Mical

$3.00 non members
Free for members with membership card

National Capitol Area User Group
P.O. Box 18088
Washington, DC 20036-8088
1-703 525 0266

Three, that MetaComco was planning a tools disk, with a pipe handler, a special kind of file re-direction, like '<' and '>' in the present CLI.

It has an AUX: device for hanging a glass teletype off the serial port, for true (?) multi-user capabilities. This tool kit disk would include other tools to improve and enhance the AmigaDOS enviroment.

(Actually, 'NEWCLI < SER:' gives this result, but this type of I/O is buffered in 512 byte blocks, which hampers interactivity.)

Four, that 1.2 was once completed, and that people attended Los Gatos "Ship it!" parties, complete with autographed Workbench disks. However, several sources claim some bugs were RE-INSTALLED to insure that Two Major Programs ran correctly under 1.2. Of course, this means Deluxe Paint and AmigaBasic.

This change was made under order from CBM West Chester. Some said that this version will be frozen in ROM in future products.

Five, that Los Gatos techies are working on a utility to read and write IBM format disks under AmigaDOS, which will breathe new life into 5 1/4 inch drive sales.

New Amiga Basic

I'd guess 1.2 will be distributed with an updated version of Amiga Basic, or a new version will be on sale before January 1, 1987. Either way, authorities tell me that Microsoft is developing an improved version, and that its specs are constantly changing. Who would need to know these specifications? Amiga Basic compiler writers.

DMCS, Deluxe Paint II

Dan Silva, author of Deluxe Paint, demonstrated Deluxe Paint II at SIGGRAPH this summer. SIGGRAPH is the graphics group of a national computer association, and their convention showcases the latest advances in computer graphics.

What can be found in the new Deluxe Paint? Virtual pictures, larger than the screen, for one. It will allow you to change resolutions without re-starting the program. Many minor enhancements, such as a patterned area fill, are also present.

Deluxe Music Construction Set is due in October, according to sources.

Former Los Gatos programmer RJ Mical, the 'father' of Intuition, is said to have signed a contract with Electronic Arts, to write an Amiga game program. Also, EA is expected to blow away SubLogic and Microprose with a new, super-fast flight simulator program, some time in the next year.

Amiga Turbo Pascal

Will the Amiga ever run Turbo Pascal? Borland promised this in the first few months of 1986. The Macintosh version was set for release in the middle of September, and inside sources said the Amiga and ST versions were to follow the Mac version.

Other sources close to the company claim version 4.0 for the IBM is a higher priority than the Amiga and ST versions. In other words, forget it for at least six months, maybe longer.

Jet and Flight Simulator

SubLogic is readying Amiga versions of Jet Flight Simulator. SubLogic employees attend their local Amiga user group meeting, and mention features of the new versions. Fortunately, the Amiga versions will be enhancements instead of straight ports.

Since the Amiga disks hold more information, more scenery will be included. Both programs will support a two-player mode, through the serial port. Beta versions used high baud rates and a direct link between two machines, but many people claim you will be able to dogfight across town, with your modem.

PC/ET emulator

Remember PC/ET, the other IBM emulator? According to Alfonso Guerra, president of SoftTeam, the alternative software IBM emulator is not ready, and may not be ready for several months.

On a message posted to several networks, he apologized for taking money before the product was ready, and offered an immediate refund to an unhappy (prospective) customers. He claimed they were surprised by the 'openness' of the Amiga architechure, whatever that means, and this hampered development.

RAM and hard disks
Many people have been waiting, waiting and waiting for Amiga hard disks and memory expansion. I suspect the market is still not settled. However, prices are dropping, and features are being added, which indicates competition is increasing.

One rumor claimed a well-known Amiga product company will bundle one megabyte of RAM with a 20 megabyte hard drive, all for $1500.

Also, several RAM board makers are promising somewhat non-volatile RAM: disks. What does this mean? It means you won't lose the files in your RAM disk when you crash or warm-boot your Amiga, as long as the crashing program didn't overwrite your RAM: disk files. The special software needed for this will determine if that memory was violated, and warn you accordingly.

This enhancement will be very attractive to software developers. They have been using memory exansions as ersatz hard disks, to speed program compilation and linking. Programs under development tend to crash alot. After a crash, reloading the RAM: disk takes several minutes.

Aegis Draw updates
Aegis Development has announced an update to Aegis Draw. This is mostly bug fixes and user suggestions. They have also announced Aegis Draw Plus, which has a slew of new features, including 640 x 400 resolution. It will be offered to Draw owners for less than difference between Draw and Draw Plus.

Bill Volk, one of the primaries at Aegis, has dropped hints that they are thinking about new drawing programs that REQUIRE more than 512K of memory. He also dropped hints they are working on a telecommunication program. Isn't everybody? How many telecom programs can the market sustain?

New products
A new company is planning several Amiga products. I promised I wouldn't reveal their name. What do you think of this line-up: A program to read and write Mac, ST, and IBM 3 1/2 disk formats, a $150 SCSI hard-disk interface, a $30 MIDI interface, and a $60 sound digitizer.

They are also busy acquiring other software for Amiga ports. They have already signed agreements with some well-known developers. They are legit, not a fly-by night group. Best of all, they have promised not to advertise until the products are READY.

PaperClip Elite
The PaperClip Elite word processor from Batteries Included will be available late this year, for $129, according to a BI newsletter.

DigiView updates
The DigiView video digitizer will have soon have new software and optional hardware. At one demonstration, NewTek showed a motorized control for the color filter wheel used to make color pictures. The unit plugs into the spare joystick port, and saves you the trouble of spinning the filter to the correct color. Also, the HAM software has been improved, to increase its ability to resolve sharp edges.

MIDI software
Soundscape and Pro MIDI, the music products from Mimetics, have finally arrived. This summer, Mimetics gave Amiga MIDI enthusiasts quite a scare when they didn't answer their phones for several weeks. They were working hard on the software, and gave the phone to the answering service.

Users complain it has too many windows, but acknowledge it as the superior Amiga MIDI product at this time. Pro MIDI program includes a MIDI patch program, and a program to convert SMUS files to Soundscape format.

More Blish books
A company called SoftCircuits produces PCLO, a circuit-design program for the Amiga. The head of the company, Ben Blish, is writing his first science fiction novel, due out this spring. His father is James Blish, a noted science fiction author.

Optical Moose
Tel-Shop, a cable TV mail order house, was selling the Amiga. The announcer said it had "an optical-digital moose", and that you could use the Amiga to pay your gas bill by opening the Utilities drawer on the Workbench.

•AC•

Finally!

ASDG, Inc. is Proud To Announce Availability of the First Intelligent Hardware Expansion Path for the Amiga™

September 22nd, ASDG, Inc. will begin shipping the Convertible™ product line as well as the Mini-Rack-B™

All Convertibles are **100% ZORRO compatible.** In fact, we've designed our board level products to work perfectly even if placed in a sub-standard ZORRO back plane (call our engineers for detailed information).

These features are common to all three:

- **ZERO Wait States - your FAST RAM will really be FAST**
- **Fully ZORRO (100 pin bus) compatible**
- **Fully Auto-Configuring**
- **Four Layer Printed Circuit Board - for quality and reliability**
- **Custom Hi-Speed DRAM Controller**
- **ONE YEAR Parts and Labor Free - in the unlikely event of board failure.**

The Mini-Rack-B (for Budget) converts most ZORRO compatible card cage cards including the Convertibles INTO SIDE MOUNTED cards. The Mini-Rack-B is a two slot card cage with a self contained 6 amp power supply. The packaging is an attractive metal card cage which does not block either mouse port.

If purchased along with a Convertible board product, the Mini-Rack-B will be available (September 22nd) for $150.00. If purchased alone the Mini-Rack-B is $300.00. The entire unit is 10 inches deep by ten inches high and only 6 inches wide.

The Mini-Rack-B coupled with our Convertible product line allows new Amiga owners the ability to start out with an inexpensive card cage and with expansion boards which will be compatible with any future upward expansion. If you had purchased ordinary side mounted products and later decided to expand to a large card cage, your side mounted products would become unusable. With the Convertibles, simply slide them out of the Mini-Rack-B and into the ZORRO back plane of **YOUR** choice.

The three memory boards are being introduced at more than a **TEN PERCENT DISCOUNT** from their list prices. Send us proof that you are a member of any Amiga User Group, and we'll take an **ADDITIONAL FIVE PERCENT** off the list.

The Convertible Memory Board can be ordered NOW:

	LIST PRICE	I INTRODUCTORY PRICE	USER GROUP PRICE
.5 Mbyte of FAST RAM:	**$450.00**	**$395.00**	**$370.00**
1 Mbyte of FAST RAM:	**$650.00**	**$575.00**	**$550.00**
2 Mbyte of FAST RAM	**$900.00**	**$795.00**	**$750.00**

Ordering Information

New Jersey Residents Please Add 6% Sales Tax
ASDG Will Pay UPS Standard Delivery In the Continental U.S.
Other means of shipment are at customer's expense.
All funds In U.S. Currency and draw upon U.S. Banks
Remember, Deliveries Begin September 22nd.
Dealer and VMR Pricing Aviailable
Demonstrations to Large User Groups Can Be Made

Send Checks or Money Orders To:

ADSG , Inc.
280 River Road, Suite #54A
Piscataway, N.J. 08854

The Amazing C Tutorial

...'C' what you are getting into.

By John Foust

A new computer language holds a particular fascination for some. It entices the weekend software buyer. It beckons the casual computer programmer, the person who programs for pleasure, not for profit.

Early Amiga developers had less choice in the matter of software development languages. Knowledge of the C language was implicitly required.

Although the developer's package included a 68000 assembler, most of the examples in the ROM Kernal manuals are written in C. The Modula-2 or assembly language afficiando must translate each example.

Of course, it can be argued that C is today's programmer Esperanto, and that Amiga-Commodore (sounds better that way, huh?) didn't have the time to translate the examples into a miriad of other languages, most of which hadn't been ported to the Amiga at the time.

Since C is the language of choice for quick and easy development on the Amiga, many novice programmers are attracted to the language. Sparks fly when a novice programmer collides with a developer-oriented compiler.

As I stated in the first Amazing C Tutorial, this column is aimed at the novice programmer, or the programmer who knows BASIC or FORTRAN, and wants to learn C. I imagine the most loyal followers of this column to be weekend programmers.

For example, today's Amiga C compilers assume some familiarity with the Unix and PC-DOS styles of command line arguments. After all, if you know C, you learned it on another computer, correct?

The compiler manuals have few examples, and they don't explain everything. They assume the reader can extrapolate from the examples given, using their past experience.

In such a foreign situation, many new C programmers are frustrated. Frankly, getting past the compiler start-up syntax is peanuts. As Hack fanatics put it: "Just wait until you get to level 28."

Don't despair. In fact, experienced programmers are often frustrated by C. I am constantly impeded by "gotchas" that take hours of debugging. Then again, at my 'real' day job, I program PC-DOS computers, which adds another level of "gotchas" beyond the standard set of C "gotchas."

Do you catch my drift? I'm telling you NOT to use C! If you are a weekend programmer, and you don't have a lot of experience in other high level and assembly languages, your progress will be stymied by C. Expect C travel to be slow. C will gleefully present you with roadblocks, one at a time. When you step over one, another will be lurking around the next bend in the code. Be prepared. Don't say I didn't warn you.

(Presumably, the weekend programmer eventually evolves into a Guru, after stepping through the ranks of local expert, hacker, wizard, et al. Sort of like the Boy Scouts.)

If you think you have exhausted the power of BASIC on another machine, you have not scratched the surface of BASIC on the Amiga. With the power of libraries, you can do a great deal in Amiga Basic. You can access most of the graphics, sound and speed of the Amiga in Amiga Basic, by calling library functions. True BASIC has its own flavor of library functions. Both BASICs allow powerful graphics manipulations much easier than C.

Most C programs call the SAME functions, so in many cases, you aren't any better off writing in C. Expect to eeeeeease into C. Don't plan on writing a magnum opus a week after you buy a C compiler.

In each language they sample, the weekend programmer faces a similar set of problems. One of the first is strings, the manipulation of text and characters.

First, I will present a set of examples in BASIC, and then discuss similar string operations in C. This C tutorial might teach you more about BASIC than about C!

Each language has its own way to process words and characters. Some languages have no special provisions for strings. C has only minimal provisions for strings.

In effect, you have to roll your own. Fortunately, every C compiler is supplied with standard library functions to manipulate a certain kind of character string.

The BASIC language is a good example of a language that has many provisions for the manipulation of strings.

It has a string data type. Any variable with a '$' character at the end of the variable name can contain characters or text.

For example, a BASIC program might give a string variable an initial value:

```
Name$ = "John Boy"
```

Unless you are a well-advanced BASIC programmer, you wouldn't know what was really happening when this statement is executed. Which bytes are getting shuffled inside the memory of the computer? Where are the bytes "John Boy" stored at any given moment?

For almost all practical BASIC programs, you can create dozens of strings, and use more of the computer's memory, without any adverse reaction from the BASIC interpreter.

This is a beauty of BASIC. You do not need to be concerned with the location or the length of character strings. In C, you have complete control over strings, with a corresponding increase in responsibility.

If a C programmer is standing nearby, and you show them this program, their mind will start to ask questions. "Where are those strings stored?" "What really happens when you enter 'Name$ = "John Boy"'?"

By this time, the C programmer will be snickering about your choice of language. C programmers look down their nose at BASIC, for several reasons, not all of them valid. Actually, the BASIC programmer has a secret load of ammunition to volley at snotty programmers, yet most don't use it.

Here's a snappy comeback: If you like to program in BASIC, and a C programmer is acting superior, ask them if they own a BASIC interpreter. Ask them if they use it. I'd wager that most do. They use it for quick and dirty programs that manipulate strings.

Although the equivalent program could be written in C, many experienced programmers still write short, quick and dirty programs in BASIC, since the interactivity and freedom of string manipulation is preferable to the edit-compile-link cycle of C programming.

Actually, on the Amiga, some C programmers are using Amiga Basic to prototype their programs. Since they can access most of the functions they will be using in their C programs, and BASIC is interactive, why not? Small user interface changes are easier to enact in BASIC than C, in many cases.

BASIC has a string concatenation operator, the '+' (plus) sign. What is string concatenation? 'Concat' means 'to put together'. Concatenation means gluing strings together, one at the end of the other.

We can continue the above BASIC example:

```
Bye$ = "Good night, " + Name$
```

After this statement is executed, 'Bye$' will contain 'Good night, John Boy'. If you remember the discussion of expressions, BASIC would refer to

```
"Good night, " + Name$
```

as a string expression. We can print this expression, or assign it to another string variable, and in either case, the value of the expression is the same.

A good question to ask is "Does this change 'Name$' in any way?" In BASIC, it does not.

In BASIC, strings are stored in a dynamic storage space. This means that the characters that make up any given string are managed by the BASIC interpreter itself, in a section of memory reserved for character strings.

This storage area is called 'dynamic' because it changes in size as your program is interpreted by the computer. As you operate on strings, the BASIC interpreter is making copies of your strings, and moving them around in memory, in the way it sees fit.

So, assuming 'Bye$' above had never been used before in our program, the BASIC interpreter will make room for it in the string storage space, and set up some other housekeeping information about the string, such as its name, and the number of characters in the string. Then, it would copy the characters "Good night, " to a temporary working area, add the characters in 'Name$' to the end, and copy the resulting "Good night, John Boy" to a freshly prepared area in the string storage space, which is associated with the 'Bye$' string.

You might have discovered this programming concept before: If you store a list of objects, you can manage the list in two ways.

One, remember how many objects are in the list, in a separate variable. Two, let the list grow as it will, but use a special place-holding value in the list to mark the end of the objects.

The first method has the advantage of allowing all values for objects. The second method has the special case object that marks the end of the list, and no list element can have that value, or the program would be confused about the actual number of objects in the list.

In the case of strings, the objects are characters, and the list is the string of characters. The BASIC interpreter uses the first method. The standard C string functions use the second.

Each character in the computer is represented by a byte, and a byte can hold 256 values, from 0 to 255. Since the BASIC interpreter remembers the length of the string, it can allow any character in the string. It could have control characters in it, such as null, the zeroth character. In BASIC, you are free to place any character in a string.

In C, the special place-holding value is the null character, character number 0. All standard C strings are terminated with a null character. In C, you cannot have the null character in a string. (Don't confuse null with the character '0', which is the printable zero. Character '0' is represented with a byte with a value of 48, in the ASCII code used in the Amiga.)

Where are C strings stored? Anywhere you want them. Usually, you would know an upper limit on the length of the strings you would be manipulating, and you would create fixed length, fixed locations for the characters that make up the string. There are other ways of storing strings in C, and those might be explored in a future column.

C does not have built-in string assignment or concatenation operators, unlike BASIC. It does have the standard library function called 'strcpy()', which copies a C string from one location to another. Since all C strings are terminated by the null character, 'strcpy()' knows when to stop copying bytes.

'strcpy()' takes two parameters, two pointers-to-characters. The first parameter points to the destination area, the second points to the characters to be moved. This order is the same as the BASIC string operator, which says 'Name$ = "John Boy"', so you might remember the order this way: "destination, source."

The function 'strcat()' is the C string concatenation function. It takes two parameters, two pointers-to-characters. The parameters are in the same order as 'strcpy()': destination, source.

To duplicate the above example:

```
main()
{
char Name[30];
char Bye[50];
    strcpy(Name,"John Boy");
    strcpy(Bye,"Good night, ");
    strcat(Bye,Name);
    puts(Bye);
}
```


This short C program, presented in its entirety, reserves space for 30 characters in a character array named 'Name', and 50 characters in an array named 'Bye'.

The characters "John Boy" are copied to the 'Name' array, in the first invocation of 'strcpy()'. Implicitly, the zero character is copied to the end, after the 'y' in 'Boy'.

(This means that you must remember to declare one more character for the null character. Also, remember that a declaration such as 'char Name[30];' reserves thirty bytes, which can be accessed as elements 'Name[0]' through 'Name[29]'. C numbers array elements from 0 to N-1, unlike FORTRAN, where the first array element is element one.)

Then the "Good night, " characters are copied to the 'Bye' array. Finally, the characters in 'Name' are concatenated on the characters in 'Bye', leaving "Good night, John Boy" in the 'Bye' array, followed by the zero character.

This example leaves no doubt as to the location of the characters in the strings of our program. Or does it? Where are the initializing strings "John Boy" and "Good night, " really stored?

When any C program is compiled, the data objects, such as the literal strings like "John Boy", are stored in the object code of

the program. In the first call to 'strcpy()', the first parameter of the 'strcpy()' function points to the 'Name' array, and the second points off into them initialized data space of the object code.

In this way, there is no doubt as to the location of the strings, and the data used to initialize them, as opposed to BASIC, where you have little to no knowledge of the location of the bytes in a string.

Here is a common mistake for beginning C programmers:

```
main()
{

char *Name;
char *Bye;

    Name = "John Boy";

    Bye = "Good night, ";
    strcat(Bye,Name);

    puts(Bye);
}
```

What happens in this example? There are two pointers to characters declared, with the same names as before. A C novice might be tempted to think a pointer-to-character is the same as a BASIC string variable.

In this example, the location of the initialized data string "John Boy" is assigned to the character pointer 'Name'. At this point, you could 'puts(Name)', and the string "John Boy" would be printed.

After the similar assignment to the 'Name' variable, the string "Good night, " could be printed, using 'puts()' or 'printf()'. This might convince you that the strings are in perfect order, and that the compiler, its authors, and the Amiga's 68000 processor itself are all out to get you, when your program Guru mediates at the next line, the call to 'strcat()'.

What happens? 'strcpy()' gleefully copies the characters pointed to by 'Name' to the location pointed to by 'Bye'. But these strings are stored in the program's initialized data space, an area generally off-limits to this sort of brutish manipulation. The characters are added to the end of the "Good night, " characters.

One problem - this area is not 'ours', in the sense of our declared data space, such as the 'Name' and 'Bye' arrays in the previous example. It may contain information important to the smooth operation of the operating system.

It may contain the storage space for other initialized data strings in our program, which will surely confuse the novice programmer, since the characters in other constant strings will appear to change under their feet.

If we add the line

```
puts("What happened?");
```

```
     John Boy
```

after the 'puts(Bye);' line, the program might work. The "What, happened?" string is initialized data, and might be stored right after the "Good night, " characters, separated by the terminating zero character of all C strings.

In this case, the "John Boy" characters might be copied over the "What happened?" characters, resulting in "ohn Boyppened?" printing, instead of "What happened?", since the 'J' was copied over the terminating zero in "Good night, ", and the "ohn Boy" string obliterated the "What ha" characters in the next string.

Since the 'puts("What happened?");' call is only composed of literal data, the 'puts()' function is passed a pointer to a specific address in memory, which is a pointer to a character, after all. This location is hard wired into the program code, because the "What happened?" string is initialized character data, and it doesn't move in memory, once the program code is loaded off the disk into memory.

This picture might help:

```
John Boy0Good night, 0What happened?0
```

I have used the printable '0' character to represent the non-printing null character, in order to show the layout of the bytes in the initialized data in the object code of our program.

In the first example, without the "What happened?" diagnostic, the end of our initialized data space was the 0 after the "Good night, " string. In the second example, the end is after "happened?".

To muddy the waters even more, imagine that an integer value is stored between the initialized data strings. If the integer, stored as two or four bytes, holds the value '1', and the 'strcat()' replaces these bytes with characters like 'ohn ', what happens to the value of this integer? Normally, it goes through the roof, and becomes very large, or very negative.

Please note that this is a somewhat theoretical example, and that if you implement it on different compilers, it may do different things, or nothing like this at all.

If you have been reading between the lines, you might realize that I am the 'novice programmer' in this example. I wasn't born with a C manual in hand, I had to stumble through this, too. I'm still stumbling, sometimes. C is a very subtle language. If you ever overhear a group of C programmers telling 'war stories', they might be telling tales of mix-ups like this example.

This is a good example of a debugging diagnostic 'fixing' a program, which can be a common and frustrating experience. A program might work well with all your debugging and testing code in place, but once you remove it, the program stops working.

The moral? If you want to manipulate strings in C, declare the space yourself, and manage it yourself. Don't be tricked by the differences between the pointers to data, and the data itself.

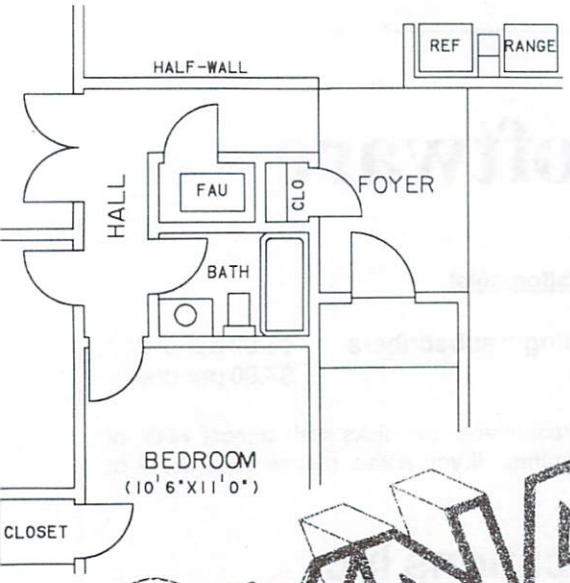
You might think that the designers of C couldn't make up their minds about C strings. After all, the implicit zero character is placed at the end of every literal string, as if in anticipation of the 'strcpy()' and 'strcat()' functions, and yet there are no built-in string operators, like BASIC's '+' and '=' string operators.

This is part of the beauty of C. The language designers knew that some data types are very common to all programs, and they added only enough support to declare data of those types, and provided only a bare minimum of arithmetic and bit-level operators for those data types. The rest was left to the C libraries, which have standard functions like 'strcpy()'.

If you don't like the way 'strcpy()' works, change it. If you want to use the '$' dollar-sign character to terminate strings, like the CP/M operating system does, then you can easily re-write 'strcpy()' to expect '$' at the end of strings. If you want unlimited characters in strings, you can write string manipulation routines similar to those in BASIC, which keep track of the number of characters in the string, instead of using a place holder character.

In C, you get exactly what you ask for. Of course, this is often not exactly what you want, which can be very frustrating.

•AC•



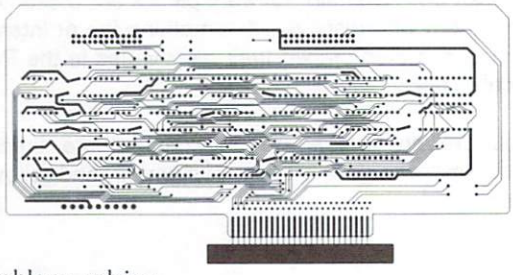
Finally, a proven CAD system for the Amiga

FROM MICRO-ILLUSIONS

DYNAMIC-CAD

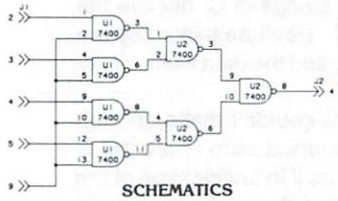
What AutoCAD* can do for the IBM*, Dynamic CAD can do with the Amiga... and a great deal more for less than a fourth of the price!

PRINTED CIRCUIT ARTWORK

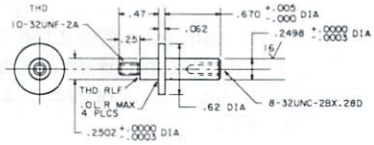


Emerging from years of successful problem solving applications in piping, and electronics for the aerospace industry, DYNAMIC CAD has brought a highly advanced and powerful CAD system together with today's most dynamic and versatile micro-computer, the Amiga. DYNAMIC CAD takes full advantage of Amiga's extensive capabilities with color, multiple modes of resolution, mouse functions, and easily accessible pull-down menus.

This is not some promised "vapor-ware." DYNAMIC CAD exists now and comes to the Amiga with a proven track record. The time and money-saving applications of DYNAMIC CAD for engineers and architects are truly astounding. Here is an advanced, 2-D drafting system with isometric capabilities that can be combined with many models of printers, plotters, and digitizers. In getting started you'll have the support of an extensive manual written in understandable English along with working examples as tutorial lessons.

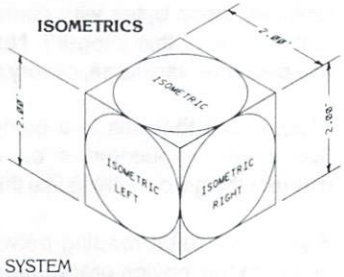


SCHEMATICS



MECHANICAL DRAFTING

ISOMETRICS



SIGNAL NAME	COMP NAME	PIN NUMB	SOURCE	TYPE COMP
S0001	J1	2	*	CON
S0001	U1	1		7400
S0002	J1	3	*	CON
S0002	U1	4		7400
S0003	J1	4	*	CON
S0003	U1	9		7400
S0004	J1	5	*	CON
S0004	U1	12		7400
S0005	J1	9	*	CON
S0005	U1	2		7400
S0005	U1	5		7400
S0005	U1	10		7400
S0005	U1	13		7400
S0006	U1	3	*	7400
S0006	U2	1		7400
S0007	U1	6	*	7400
S0007	U2	2		7400
S0008	U1	8	*	7400
S0008	U2	4		7400
S0009	U1	11	*	7400
S0009	U2	5		7400
S0010	U2	3	*	7400
S0010	U2	9		7400
S0011	U2	6	*	7400
S0011	U2	10		7400
S0012	U2	8	*	7400
S0012	J2	4		CON

NET LIST FROM SCHEMATIC

WHAT DYNAMIC-CAD CAN DO FOR YOU

- D·C gives you all the expected CAD functions of zooming, rotating, panning, group functions and menu driven features.
- D·C brings you professional CAD capability tested and proven in the production of tens of thousands of drawings.
- D·C will liberate you from the need to draw free hand.
- D·C has net listing capability from your schematic.
- Schematic comparison to your printed circuit artwork for continuity check.
- D·C can produce isometric views.
- Mil-Spec quality Leroy* fonts.
- Automatic line dimensioning.
- D·C includes a series of information libraries: Symbols, Electronic Parts/Chips, Architectural Components, Landscaping, etc.
- Data base to store and retrieve information on parts specifications, vendors, and pricing.
- Data base system utilizes ASCII format files which are convertable to other standards.
- Capable of utilizing up to 4,096 colors.
- D·C can generate over 8,000 layers.
- D·C supports most standard dot matrix printers, ink jet, lazer jet, pen plotters, and the Gerber* Photoplotter.

SYSTEM REQUIREMENTS
512 K RAM
2 Disk Drives (or)
1 Drive and Hard Disk
Printer or Plotter

Inquiries invited. (818) 360-3715

*Gerber — trade mark of Gerber Scientific Instruments
*Leroy — trade mark of Keuffel & Esser
*IBM — trade mark of International Business Machines
*AutoCAD — trade mark of Autodesk, Inc.
*Amiga — trade mark of Commodore Business Machines

Micro-Illusions

P.O. BOX 3475, GRANADA HILLS, CA 91344

Miga-Mania!...

Screen SaVer

Don't allow your monitor's phosphors to "burn in"

By Perry Kivolowitz
Usenet: ihnp4!ptsfa!well!perry

Hey! This month represents the first issue of Amazing Computing to contain the new format Miga-Mania! Under the new format, I will present a short (three hundred or so lines) program each month and explain how the program was written and how it works. Every month you'll get a useful program plus a tutorial on how to juice up your software library, as well as your knowledge base with each copy of Amazing Computing you acquire!

I'll begin with a program which I've heard many people wish existed. There is a need for a small program which would detect extended periods of "user" idleness and, if such a period is detected, cause the contents of the screen to be replaced with some image which would not allow the monitor's phosphors to "burn in."

As you might know, some computer terminals have this feature built into their firmware. That is, if no key is struck for a given period of time, the screen is blanked out. To return the video signal to the screen, the user simply hits any key.

This is an important feature since monitors may be damaged if they are asked to display a static (non-changing) view for extended periods of time. If you've ever seen a "burned" monitor, you'll have noticed that the screen contains an image even when off. What you're actually looking at (the ghosted image) are the damaged phosphors.

Detecting extended periods of "user" idleness on a computer terminal is pretty straight forward. Just keep track of the elapsed time since the last time the user struck any key.

On the Amiga detecting periods of "user" idleness is not easy. First, why do I keep saying "user" idleness? Why not invoke the screen saver based upon c.p.u. idle time? Let's say you had a graphics program that spent a huge amount of time computing before it rendered its result into a window. While the computation takes place, the machine is, of course, not idle and therefore the screen saver won't kick in. However, if you go to lunch while your Amiga chugs away, your monitor displays the same unchanging view for a potentially long time.

Ideally we want to simulate as closely as possible the situation with computer terminals. That is, blank the screen only after the user has indicated to us that he's no longer around to look at the screen. To do this we have to try to figure out when the last time the user struck a key or moved the mouse.

Ummmm. Tricky. Tricky because mouse and keyboard activity are being handled by whatever program the user is running, not us. There are, however, two ways this could be done.

Both methods make use of the fact that AmigaDos starts a task whenever a CON: device is opened or whenever Intuition is running. The task, called the "input.device" acts as a conduit for keyboard, mouse, and timer i/o. The input device, for example, is the program which moves the mouse pointer around. It's what Intuition uses to pass keyboard or mouse activity back to your programs using the IDCMP (Intuition Direct Communications Message Port).

So, we can measure user idleness by detecting activity by the input device (if for some rare set of circumstances the input device is not running, which does happen if you use programs which take direct control of the machine, we'll have to use some other metric for deciding when the user's been idle).

The first way of detecting user activity through the input device is by inserting a small monitoring task into the list of input event "handlers." Input event handlers are a list of routines to which the input device passes all keyboard or mouse activity. The input device basically runs down the list of handlers asking: "Does this make sense to you?"

Each handler has an opportunity to accept the input event and claim it for its own use thus removing it from consideration by the remaining input event handlers. This is how, for example, the close gadget is intercepted by Intuition and passed to you as a close gadget selection rather than as a normal mouse button depression.

Instead of claiming the input event for itself an input handler can simply ignore the input event allowing it to be passed on to the other input handlers. Or, the input handler can record the nature of the event before it passes it on to the other handlers. That is, insert an input handler into the input device at the very front of the input handler list. Every time an input event comes along, simply note the event and pass it on. If a long enough period elapses without seeing any input events, kick in the screen saver.

Though conceptually simple, it's not the method of choice for two reasons. First, coding wise, it (insertion into the input handler list) is pretty complicated and I didn't feel like figuring out how to do it. Second, the code to do all this would take up a fair amount of memory. Remember, we want this program to be started from the startup-sequence and hang around all the time. As such we don't want to waste more memory than necessary.

The second method of detecting input device activity (and thus user activity) is pretty novel and very easy to implement. I said to me self, I says (sorry, I'll turn off Popeye now) "If the user is

not doing anything, then the input device must be waiting at some instruction for an event to happen just like a user process would wait for an Intuition event." If this is so, we should be able to detect this by examining the program counter of the input device task. Woah.

The program counter is the machine register which tells the computer which instruction is the next one to be executed. When a task is bumped from the c.p.u. so that another task may run (remember that the Amiga is a multi-tasking computer) the complete state of the bumped task must be saved. This includes all the machine registers among which the p.c. (program counter) can be found.

When a task is bumped from the c.p.u. the Amiga EXEC pushes the machine registers of the bumped task onto the task's own stack. This, by the way, accounts for the minimum allowable stack size of 70 bytes. That's how much space is required to hold all the saved machine registers for a given task.

The location of the task's stack is readily available from the task structure whose location can trivially be found (thanks Amiga!) by calling FindTask.

So, detecting input device activity is as simple as examining a memory location at a specific offset from the input device's current stack pointer.To be honest I really have no idea at what offset from the stack pointer the p.c. is saved.

To be very, very honest I really have no idea what it is I am actually looking at (two long words behind the current stack position) but it seems to do exactly what I want it to do so.....

In keeping with the idea that the screen saver should occupy as little memory as possible, I originally attempted to use the graphics macros ON_DISPLAY and OFF_DISPLAY which, I thought, would interrupt the flow of data to the screen and thus afford a completely blank screen.

The advantage of blanking the screen this way, would be to make the program exceedingly small if constructed using these macros. Also, the screen would be rendered completely blank, in fact I had hoped that the OFF_DISPLAY'ed screen would be as dark as a powered down monitor (only the power on indicator would tell you the monitor was even on).

The disadvantages of using OFF_DISPLAY and ON_DISPLAY were two. First, a completely blank screen is exceedingly boring. The other reason was that after getting the macros to compile properly (with Manx 3.20A), I detected no useful action caused by OFF_DISPLAY so there seemed to be little use in investigating this avenue.

I decided to construct a pretty lively display in which no phosphor would be overly taxed. The display would be interesting to look at while at the same time, designed not to excite any particular phosphor for any extended period.

When the screen saver kicked in, I brought a one bit plane screen to the front. The screen is allocated dynamically and does not take up any memory until the screen saver has actually detected an extended idle period.
In choosing a one bit plane screen too much memory will not be consumed and any drawing operations performed will take place with little effort.

Then, rising from the dark screen came one hundred or so single pixels. The dots are of a randomly chosen color with the proviso that the brightest color values the Amiga is capable of are excluded (a dimmer pixel is a less excited pixel, hi ho).

Along with the dots will appear a happy face 21 pixels high by 37 pixels wide. More about the happy face later.

After all the dots and the happy face have assumed their ultimate color values, they begin to dance merrily about the screen. Each dot follows its own course at its own speed. The happy face also moves around the screen but with a predictable course and speed.

After a while the dots will begin leaving a trail over where they've been. The happy face will appear to float behind the trail. The screen will become covered (for a short time) with dot trails and the user may become totally mesmerized.

Then, suddenly, everything stopped and then slowly faded away. Whatever screen was up front when the screen saver kicked in is brought to the front again for five seconds at the end of which time the whole cycle will repeat itself (with a different color and different dot tracks).

Now, you might ask, what's the happy face for? Well, to answer this question let me tell you the two ways you can "exit" the screen saver.

If you click the left mouse button (LMB) anywhere on the screen but NOT on the happy face, the screen saver will go back to its dormant state of periodically checking for an extended idle period. All of the memory used by the screen with its dots will be returned to the free pool.

If you click the LMB DIRECTLY ON the happy face (you gotta be fast!) the screen saver will actually exit in the formal sense. That is, the program will return all of its resources and then terminate completely.

The happy face is actually a full fledged Intuition gadget that I have set in motion! The imagery itself was created using Deluxe Paint from ElectronicArts. I saved the face as a DPaint BRUSH and used the program GI to convert the saved brush file into C statements suitable for inclusion into a program. The program GI is available on Fish Disk #14 and was written by Mike Farren. Thanks Mike and Fred (how's the new place?).

Before we get into the code, one more thing worth mentioning. While talking to Gary Murakami (a person of much UNIX (TM) fame, now an Amiga enthusiast), he mentioned that people ought to be able to specify how long a time period is to be considered "really" idle. Some people might want one minute, others might want twenty. So, at Gary's suggestion I added a command line argument (which means the screen saver must be run from the CLI and, in fact, would be a natural contender for inclusion into the start up-script)which specifies how many idle minutes are to elapse before the screen saver kicks in.
If no time period is specified a default of four minutes is used. Now! On to the code!

I declared some routines to be external to this module. These routines will in fact be defined by the C compiler libraries during program linkage. I have to declare these routines because if I didn't the C compiler would not have any way to tell their type at the time the program was compiled.

Remember from an earlier tutorial I wrote in Amazing Computing, those functions which return integers don't have to be explicitly declared as doing so but functions returning any thing else must be explicity declared so that the function's type can be determined accurately.

Next, I declare some preprocesser symbols which I find useful to cut down the amount of typing I have to do later on. One of the hassles in writing well structured code is that sometimes, due to indentation, lines become exceedingly long. These preprocessor symbols also often help cut down on long line lengths.

One symbol worth noting is GMEM. GMEM is defined as the logical 'or'ing of two other preprocessor symbols (defined in exec/memory.h). MEMF_CHIP means that the memory about to be allocated should come from chip memory and thus be accessible by the graphics coprocessor.

Since GMEM would have been used here, why not throw in MEMF_CLEAR which says that when the memory is allocated it should be initialized to zeroes. Why not!

NPTS defines the number of dots I'm going to move around the screen. HAPPY_FACE is a symbol standing for a sentinel value. If I see this value come back from the "zoom" routine, I know that the user struck the happy face gadget and wants the screen saver to terminate.

FACE_WIDTH and FACE_HEIGHT are, of course, the width and height of the happy face gadget. Notice I cast them to longs by appending an "L" to the digits.

Next I undefine the symbol "min". I have no idea at this point if the symbol has been defined by someone somewhere and if it has, I have no way of knowing how. So, I deep six it if it was there. If it wasn't already defined then there's no harm done.

I define min to be a preprocessor function returning the smaller of its two arguments. Notice the use of the trinary operator "?". The value of the whole expression will be the value of the sub-expression following the question mark up to the colon if the value of the sub-expression appearing before the question mark evaluates to TRUE (non-zero). If the initial sub-expression evaluates to FALSE (zero) the value of the whole expression becomes the value of the sub-expression following the colon.

Think of the trinary operator "?" as a way of coding "in-line" IF statements.

Notice that I parenthesize everything, even each instance of the macro's arguments. Heavy parenthesization is needed to prevent some pretty nasty bugs from cropping up.

For example, if I wanted a macro which returned the square of its argument I would write:

```
#define SQUARE(X)((X) * (X))
```

and not:

```
#defineSQUARE(X)X * X
```

Why? Let's say the argument passed was "2 + 5". The expected result is of course 7 squared which is 49. If we passed "2 + 5" to the second form of the macro we'd get "2 + 5 * 2 + 5" which is 17.

Next comes the most important line of the program which if removed would cause an immediate and irrecoverable (ever) system crash. That is, my name. All kidding aside, always give credit where credit is due.

After the author's credit, I define the strings which defined which libraries we are going to open. In this case I will be using the graphics library as well as Intuition so both these libraries must be named.

Correspondingly I need to declare library base pointers for each library I will later open. Next I declare some pointers to entities I'll use later as well. These include pointers to the screen and window I setup as well as copies of certain pointers that the window and screen structure contain.

I provide a place to copy the ViewPort pointer, a RastPort pointer and a BitMap pointer. The BitMap is used to reference the happy face image during the blitting operations which send the face to the screen. The ViewPort and RastPort pointers are copied so that when they are used they can be readily accessed instead of incurring the costs of dereferencing a window or screen pointer.

One note about the happy face imagery. Normally when I load a pre-made image into a program I set up code to AllocRaster a duplicately sized array from CHIP memory and copy the original image to the newly allocated memory.This has the advantage of ensuring that the image data will reside in CHIPmemory. The very severe disadvantage is that I double the memory requirements of the image.

After all, the initialized array of unsigned short integers is going to sit in memory as well as the dynamically declared CHIP memory buffer. The way I implemented this program, the initialized data (only a couple of hundred bytes) must reside in CHIP memory when the program is loaded. With MANX C this can easily be done by specifying the loader flag "+c" along with the specifier for initialized data.]

Note that the BitMap variable 'fbm" must also go in CHIP memory. Since I have to use the Manx ``+c" construct anyway, I can force fbm to also go into CHIP memory by declaring it as a static. This way the compiler treats the BitMap variable in the same way (with respect to memory placement) as the initialized array of shorts defining the happy face.
If you have Lattice C the ATOM utility can be used to ensure that the data is loaded properly.

Next I declare sleepy_time to the default value of four minutes. The screen saver will check for inactivity every ten seconds or six times a minute.The initializer can therefore be read as four minutes at six times per minute.

Each dot drawn on the screen will be controlled by a "point" structure. The x and y fields keep track of the dots current position. The dx and dy fields represent the amount in the x and y directions the dot will move each time it is updated. These values are initialized to random values within given constraints.

Next I declare the happy face gadget structure. Notice that I select no high lighting and no imagery as well. It's as if I want Intuition to make believe there was a dead zone somewhere on the screen which was actually a gadget. This is ,in fact, what I want. I'll take care of rendering the gadget's imagery within the gadget's selectable area since I can do it a lot more efficiently than constantly calling RefreshGadgets.

The NewScreen struct specifies that Intuition should set up a 640 by 200 single bit plane screen. I ask for defaults in font and color since these initial values don't mean anything to me.

The NewWindow structure specifies that the newly created window will occupy the entire screen since it is a full screen borderless backdrop window. Backdrop windows are created behind all other windows (hence their name). Sometimes more than one application that you might wish to run at the same time will declare a full screen backdrop borderless window. This has the obvious problem of succesive backdrop windows being forever obscured by older ones. I don't have any fear of using a backdrop window here since this will be the only window rendered on the screen.

Notice that I want to be told of any mouse depression. Any LMB push not on a gadget will be sent due to MOUSEBUTTONS. Any LMB over a gadget (there is only one) will be sent due to GADGETDOWN (which by the way is the proper counter part to a gadget specifying GADGIMMEDIATE). If somehow the LMB is depressed over someone else's window or screen entirely I'll STILL be notified because of INACTIVEWINDOW. As you can see, I'm pretty serious about intercepting any LMB action since this is the way the user exits the screen saver and resumes gainful human/computer interaction.

Next comes the definition of the happy face. Since this data will be pointed to by a BitMap structure directly and thus addressed by the custom display hardware of the Amiga, this data structure must reside in the first 512 Kbytes of Amiga memory. As mentioned before this is easily accomplished with either Manx C or Lattice C (or by not having any extra memory on your machine) By the way, speaking of extra memory, did you hear that benchmarks on the CardCo "FAST" RAM expansion show that the "FAST" RAM slows your machine down by 32 percent? (*source--Net News NewsGroup: net.micro.amiga* Nice design fellas! Talk about recognition from your peers).

That at last starts the actual code of the screen saver. The main routine (named main...see some things in C language are intuitive!) is very straight forward. Note the use of argc and argv. The way I use argc and argv indicate (correctly) that this program expects to be run from the CLI since it looks for an ASCII argument specifying the number of minutes of idle time needed before the screen saver kicks in. However, should the screen saver be run from the Workbench no harm will be done since the incoming argc will be zero causing the screen saver to default to four minutes.

Let's get the libraries we need opened up. Notice the use of the preprocessor symbols to cut down on the number of characters on the line (IBPtrand GBPtr). Also, note the zeroes as the second argument to OpenLibrary. The zero specified where the version number is usually found indicates to EXEC that we really don't care which version of the libraries are out there. Just open'em up!

The call to init_happy_face does some one time only chores which I described above. That is, I call InitBitMap on the BitMap structure which will look after the happy face and also make the BitMap point to the happy face imagery data.

If either of these two libraries had been disk based I would have delayed their opening until I had identified an extended idle period. I would do this, of course, to reduce the amount of memory used by the screen saver all the time. But, since both libraries are already in RAM it doesn't matter when I open them.

The following IF statement ensures that I will enter the main loop of the program only if both library openings had been successful.

Next we drop into the loop where I'll remain until the user actually picks on the happy face with the LMB. Result is the value passed back by the "zoom" routine which indicates why the zoom routine returned. If the user had delibrately brought the LMB crashing down over the poor happy face, zoom indicates that the user wishes to fully terminate the program by passing back the value HAPPY_FACE.

OK, here we are. We're in the central loop. The first thing I do is wait for inactivity. The wait_for_inactivity routine will not return until some number of minutes of complete keyboard and mouse inactivity has passed.

When wait_for_inactivity returns, the user has gone off to meditate. NOW we'll allocate the screen and window. This way a minimum of memory is used by the screen saver while it is monitoring user activity. Only when the screen and window are needed do we call them into existance ("You read a scroll of createwindow. --more-- A window is created").

S this convention and set color register zero to the darkest value possible (0 red intensity, 0 green intensity, and 0 blue intensity).

I also set color register 1 to all zeros as well. Curious? Doesn't this mean that if I draw stuff with color register 1 it will appear the same as stuff drawn with color register 0? Absolutely.

Remember, colors are not written into the display memory. Color register numbers are. In the display memory the background pixels will have a value of zero. Pixels that I draw with the foreground pen (notice the coming SetAPen to color register one) will have a value of one. Thus in the display memory, background pixels are quite distinct from image pixels.

However, temporarily (at least) color registers 0 (the background) and 1 (the image) are mapped to the same colors so that I can busily draw the image without the user seeing what I am doing (since to him/her all the pixels look the same).

Anyway, after setting the two color registers to the same dark value I initialize the entire window by calling SetRast and specifying which color register should be used to fill the entire window.

I set the drawing mode to COMPLEMENT so that anything I write will be exclusive or'ed into what's already there. This is really useful for cheap animation. Exclusive Or works like this:

Writing	On Top Of	Results In
0	0	0
0	1	1
1	0	1
1	1	0

Thus if both operands are equal the result is zero. If the operands don't equal each other the result is one. The really neat part of the above table is the last two lines.

These specify that if I Exclusive Or an image into blank memory the result will be a direct copy of the image. That's because I Exclusive Or'ed a bunch of ones (in the image) with a bunch of zeros (in the blank memory). If I wanted to erase the image I simply repeat exactly the operation that drew the image in the first place.

That is, I'll Exclusive Or a bunch of ones (in the image) with another bunch of ones (placed in memory by the last operation) to produce a bunch of zeros. Thus Exclusive Or is its own inverse operation.

So to animate a dot across the screen I simply:

1 Write the initial dot at x,y
2 Exclusive Or dot at x,y (erase)

3 x += x_increment;
y += y_increment;

4 Exclusive Or dot at x,y (draw)
5 Goto 2

Returning from set_color_map I enter the loop in which we'll stay as long as the left mouse button is not depressed. In this loop I bring the newly created screen to the front by calling ScreenToFront. Of course, the first time through the loop the newly created screen will already be the front most screen. This call is needed because at the bottom of the loop I call ScreenToBack which causes whichever screen was front most when the screen saver kicked in to once again be the front most screen.

The original front most screen stays up front for five seconds and then is replaced by the screen saver screen when we loop back to the ScreenToFront call.

After moving the screen saver screen to the front I call the zoom routine. Perhaps it is misleading to call the routine "zoom." I did so because it makes stuff "zoom" around the screen (one might think that it "zoomed in or out" or something).

The zoom routine will return non-zero if the while (1) loop needs to be broken. If the user hits the LMB anywhere on the screen (but not on the happy face) a value of one is returned causing the break statement to be executed. If one is returned, notice that the surrounding while loop (result != HAPPY_FACE) is not broken thus we loop back to waiting for an extended idle period.

If the user selected the happy face with the LMB then zoom will return HAPPY_FACE. HAPPY_FACE being non-zero will break this inner loop and will also break the outer loop explicitly causing the end of the program.

Let's get into the zoom function now.

Class is declared as a place to save the Class value which is part of every IntuiMessage. Recall from my eariler tutorials that if I want to refer to any part of a message after I have replied to the message, I must explicitly save the message contents somewhere outside the message. This is because once I reply to the message I can no longer be sure of the contents of the message since Intuition is now free to reuse the message's memory.

I need to remember the Class of the message so that I can detect GADGETDOWN messages which can only be caused by the user depressing the LMB over the happy face. (You know, I wouldn't want to be a mouse button. They are always being struck or depressed!)

I declare a register based pointer to a structure of type "point". I am going to use this pointer a great deal so it pays to put it into a register. Specifically I will use this register based pointer to step through the array declared on the next line.

I declare the array of point structures which will be used to look after the dots on the screen. Notice that by declaring the array (which is fairly large) here it gets declared off of the program's run time stack. My thinking was that this is exactly what I want. The memory used by the array will not be allocated unless it is needed. Then I remembered that the Amiga pre-allocates each process stack anyway so this advantage is negated.

Likewise I declare a point structure to look after the happy face as it travels around the screen.

Another register based pointer is declared to help me step through the array of "point" structures. I initialize the pointer to the address of the next location beyond the array. As I step through the array I can test p against this boundary pointer to tell when I have stepped throught the entire array. Done this way, very little overhead is produced (ie: I needn't have a secondary loop counter).

I next declare in long integers a place to put a red, green, and blue color value. Since these numbers range in value from 0 to 15 they obviously don't need to be in long integers. However, when they are passed to EXEC in SetRGB4 calls, they are passed as long so why not avoid generating code to convert the 16 bit Manx integers into 32 bit long integers by declaring them as longs in the first place?

The style variable is used to switch between erasing the dot's trail and not erasing the dot's trail. One pass through the zoom routine is comprised of moving the dots around for a while erasing their trail as I go followed by a period of moving the dots around but leaving their trail intact.

Loops is used to count the number of times through the main zoom loop I have gone. Because of the WaitBOVP call I have in the main loop, it will take at least the time of one video frame for each loop to complete. More on WaitBOVP later.

Finally, I declare an IntuiMessage pointer to receive the results of calls to GetMsg.

The first thing I do is set up the dots on the screen by calling set_dots passing it the base address of the list of point structures.

Set_dots is a simple routine which for each structure (dot) choses a random place on the screen for the dot to initially be and also chooses a random delta in the x and y directions. As the x or y deltas become larger the speed of the dot increases. If it works out that there is a large difference between the x and y deltas then the slope of the dot's movement will be far from the diagonal. In this version of the program I have set the range of delta x's and y's to be between -6 and +6.

Notice the use of a C language DO loop. A DO loop is similar to a WHILE loop except that the body of the loop is executed at least once since the end-of-loop conditional is evaluated at the end of the loop rather than at its beginning.

In natural language the intent of the loop is the following:

Keep assigning random values while the last values chosen are no good.

Notice that this is different from a WHILE loop which would be: Assign a first set of values; While the values assigned are no good, keep assigning new ones.

Once I've gotten good placement and have initialized the variables affecting the dot's movement I call WritePixel to draw the dot using color register 1. Color register 1 will be used since I called SetAPen specifying that register. The newly drawn dot will not be visible at this time however because I specified that color register 1 should contain the same color as the background.

Remember, you don't write colors into the display memory. You write color register numbers.

Next I initialize the position and movement of the happy face by calling set_face passing it a pointer to the point structure used to look after the face.

I chose a random x and y coordinate for the upper left hand corner of the face in such a way as to ensure that no part of the face will extend off any edge of the screen. Rand(580) returns a number between 0 and 579 to which I add 20 making a number between 20 and 599. Since the face width is 37(FACE_WIDTH) even if the random number is as large as it can be I still fit inside 640. The y coordinate is chosen similarly.

The two starting coordinates are loaded into the x and y position fields of the point structure. They are loaded into the happy face gadget structure as well to become the position of the upper left hand corner of the invisible gadget's select box. And that's the secret to the moveable full-fledged Intuition gadget.

This works because the input.device intercepts the LMB event first. It passes the coordinates of the mouse down its chain of input.device.eventhandlers. When it gets to Intuition's entry in the list, Intuition simply compares the LeftEdge, TopEdge, Width, and Height of all your known gadgets to the current mouse position. If the mouse position is inside one of your gadgets, Intuition tells us which one. Thus I could happily change the LeftEdge and TopEdge and thereby move where Intuition thinks the gadget is.

Next I set up a standard delta x and y. I didn't make this random since I wanted the happy face to move slowly enough to make it easy to hit with the LMB.

Now I draw the happy face in its initial position by calling BltBitMap. This routine is similar to ClipBlt but does not handle clipping the source to the target so is a little faster. Notice the 0x60L this specifies the blitting operation should be exclusive-or. I have discussed ClipBit in a previous tutorial, and will go into it in more detail in next month's Miga-Mania.

Returning to the zoom routine, the next thing I do is compute three random values specifying what final color color register zero should map to.

Having set up the final values of red, green, and blue, I call ramp_up to slowly bring the value of color register one up to match the chosen values.Ramp_up and ramp_down are very simple and, as you would expect, quite similar.

In ramp_up I loop through the possible values of each color component and load into color register one the smaller of the current loop value and the desired value. Thus if the final RGB value was 10,5,7 the values loaded into color register one by ramp_up would be:

Iteration	Red	Green	Blue
0	0	0	0
1	1	1	1
2	2	2	2
3	3	3	3
4	4	4	4
5	5	4	5
6	6	4	6
7	7	4	7
8	8	4	7
9	9	4	7
10	10	4	7

and so on...

Back in zoom, I'm ready to enter the outer zooming loop. This loop will be executed twice before we automatically return to main. I control the loop with the variable "style." If style is zero then we animate the dots over the screen. During the second time through the outer zoom loop style will be one causing the trails made by dots in their travels not to be erased.

For style zero (erasing trails), I set loops to 3500. This choice is purely esthetic. Loops will count down towards zero terminating the middle zoom loop when zero is reached. Notice I coded:


```
while (--loops) {
```

rather than:

```
while (loops--) {
```

or

```
while (loops-- >0) {
```

It is interesting to note that the third of the three (basically) equivalent forms requires more c.p.u. cycles to execute. In Manx C, if I had placed loops into a register, the first form translates into:

```
sub.w#1,d4
tst.wd4
beq.3
```

While the second form translates into:

```
move.wd4,d3
sub.w#1,d4
tst.wd3
beq.5
```

And the third into:

```
move.wd4,d3
sub.w#1,d4
cmp.w#0,d3
beq.7
```

Let's look at the last two forms:

Given that the move and sub statements are the same either way, let's look closely at tst compared with cmp.

	Instruction time for operating on a word	Instruction time for operating on a long
tst	4	4
cmp	8	14

This means that as coded the cmp takes twice as long as the tst. If I had requested Manx to use all 32 bit integers or had put the loop counter in a long, the cmp would take 3.5 times as long. The reason for this is that cmp is a lot more powerful then tst.

My purpose was to compare against zero. This is exactly what tst does, compare to zero. Cmp compares any two arbitrary values. By saying:

```
while (loop--) {
```

We are testing against zero implicitly. While if (if while) we say:

```
while (loop-- > 0)
```

the compiler doesn't look ahead far enough to see that what's being tested against happens to be zero but rather goes ahead and generates the general case code.

Why bring this up? Well, in a small loop the overhead in administrating the loop can sometimes dominate the cost of the loop. If the entire loop takes 20 clock cycles to execute and you execute it say 100,000 times, you can eliminate four clock cycles per loop and save 400,000 cycles!

Now, look at the first form. I can eliminate the move instruction entirely by changing the order in which the decrement is done. If I post-decrement the compiler has to keep the old value around to compare against. However if I pre-decrement the subtraction and test against zero can be made against the same location.

Whenever you post-decrement (or post-increment) you can probably change the code slightly to make it a pre-decrement (or pre-increment) saving an entire instructiion. Since the pre or post increment/decrement construct is often found in loops, the savings is multiplied substantially.

Getting into the loop, the first thing I do is check to see if the IDCMP has a message for me. If so, the IF statement will evaluate to TRUE because the value returned frlom GetMsg into message will be non-zero.

If there is a message, I quickly save the message class for later use and reply to the message. Then executing the BREAK instruction, I exit the middle zoom loop returning to the outer zoom loop (where I will exit that as well). An IDCMP message can only mean the user wants to get back in the saddle. At the end of the zoom routine I look at the saved value of the message class to see what to return to main.

Given that there are no IDCMP messages waiting for me I get down to business. The pointer variable I will use to step through the array is set to either the first, second, third, or fourth element in the array. I do this because in any given loop I am going to move only one quarter of the dots. Why? Moving only 25 percent of the dots in any given video frame slows the display down to a comfortable pace. I emphasize that this is not a hardware based restriction but rather an esthetic decision.

(This is similar to people commenting that the bouncing ball demo made the ball bounce so slowly. Was there something wrong with the machine? No. It was purposefully slowed down so that it would be fun to watch. But, of course, Atari said....but I digress.)

Every eighth video frame I, want to change the color of the mousepointer slightly so that the mousepointer, while still being visible, does not itself burn the screen.

Notice the way I avoided using the modulus ("%") operator in C. The modulus operator cannot be executed directly on the 68000 and must be implemented in software. Since I want to take the modulo of a power of two I can achieve the desired result by logically anding with one less than the desired modulus.

For example:

```
23 % 8 = 7 but 23 is 10111 in binary
so 10111 & 111 = 111 or 7.
```

Jerk_pointer selects one of the color registers reserved for sprite zero (the mouse pointer) and randomly changes it.

Next I wait for the video beam to reach the bottom of the screen. After reaching the bottom of the screen there is a small amount of time (the verticle retrace time) in which I can do anything I want to the display without letting the user see any intermediate portion.

A key issue when doing any kind of animation is synchronizing with the video beam. Changing the display memory while the video beam is loading pixels from the display memory being changed will produce a video frame which is partially the old scene and partially the new. The human eye can detect this sort of mistake very, very easily.

Not being synchronized with the video beam while moving the dots around, however, won't be a problem since each dot is only one pixel, there's no way to screw things up. But...the happy face should definitely be drawn when I'm sure the video beam won't sweep over the disply memory being affected.

Therefore I call WaitBOVP (Wait for Bottom Of View Port) to get me in sync with the video beam before I move the happy face.

Move_dots given a 0 as its first argument, will move the happy face, while passing a 1 as the first argument, tells move_dots that what we're moving this time around is a dot.

Move_dots is passed the style value so that if what we are moving now is a dot we can decide if we want to keep or erase its trail. The third argument specifies a pointer to a "point" structure which defines how and where to move the object in question. In move_dots first I save the old x and y positions of the object. Then I offset the x and y positions by their respective deltas. If moving the object should cause them to move off screen, I undo the move and alter the sign of the delta causing the object to move off the screen. This implements the bouncing off the sides of the screen effect.

What defines the edge of the screen depends on if we are moving a dot or the happy face. If we're moving a dot then the edge of the screen in the x dimension is 0 on the left and the screen width on the right. For y it is 0 on the top and the screen height on the bottom.

If we are moving the happy face the right and bottom dimensions are decreased by the happy face width and height respectively. If we are moving a dot then erase the dot at its former position if style is not zero. In any event draw the dot at its new position. If we are moving the happy face then redraw the happy face at its previous position (causing its image to become erased, remember how Exclusive Or works?). Then draw the happy face at its new position. Also, remember to alter Intuition's idea of where the happy face gadget is located.

Now it's time to move dots. As we step through this loop we advance the point structure pointer by four structures. Remember that the initial value of the point structure pointer cycles around each of the first four structures in the array. Thus, I've partitioned the array into four equal parts, one of which I will update for each video frame.

When the three loops have completed or the user has depressed the LMB we get to the ramp_down call. This call undoes the affect of ramp_up. We then set the entire display to color register zero thereby initializing it for another go around.

Now to decide what value to return to main. If message is NULL no message was received thus the user is still out to lunch. I return a zero which causes main to immediately re-enter zoom.

If the user did depress the LMB (thereby causing a message) the message can either be for a gadget down or not. If it's for a gadget down then report back to main with a special value signifying that main should close down the program. If the message was for any other reason pass back a one which causes main to go back to looking for idle times.

That basically completes the discussion of the screen saver (which I call ssv for Screen SaVer).

The way I implemented the idle period search, while being very simple, does have a major drawback. That is, the mouse has to be in motion or a key has to be in the act of being depressed when InputActivity checks otherwise the activity will be missed.

I suggest that you start ssv from the df0:s/startup-sequence file. Like so:

```
run ssv 20; assumes ssv is in c:
; will wait 20 minutes before kicking in
```

See you next month!

•AC•

```
#include <exec/types.h>
#include <intuition/intuition.h>
#include <graphics/gfxbase.h>
#include <exec/memory.h>
#include <exec/execbase.h>

/*
**      s s v   -   a   S c r e e n   S a V e r
**
**      Copyright (C) 1986 By Perry S. Kivolowitz
**
* The author grants permission for the non-commercial
* distribution of this software provided that this
* and other indentifying information remains intact.
**
* This software is provided ``as-is'' and carries
* with it no explicit or implicit promise of support
* by the author.  Nor will the author be held liable
* for any damages,  real or imagined, which may
* result from the use or abuse of this software.
**
*/

/*
* This program is intended  to  be  RUN from the
* startup- sequence. It occupy a small amount of
* memory and comsume  very few c.p.u. cycles. It will
* attempt  to  detect some  number  of  consecutive
* minutes in which you haven't done anything at the
* keyboard or mouse.
**
* After some minutes, ssv  will  bring a dark screen
* to the front and do some eye magic designed not
* to hurt your monitor. From time to time ssv will
* bring  your original screen to the front again just
* to remind you what's running.
**
* Clicking the left  mouse  button  at  any time
* causes ssv to delete its screen and go back to
* waiting for inactivity. If you want ssv to actually
* EXIT - then click  on  the happy face which will
* float around the screen.
**
*/

extern void *OpenLibrary();
extern struct Screen *OpenScreen();
extern struct Window *OpenWindow();
extern struct IntuiMessage *GetMsg();
extern struct Task *FindTask();

#define RPPtr struct RastPort *
#define BMPtr struct BitMap *
#define IBPtr struct IntuitionBase *
#define GBPtr struct GfxBase *
#define GMEM (MEMF_CHIP | MEMF_CLEAR)
#define NPTS 100
#define HAPPY_FACE 999
#define FACE_WIDTH 37L
#define FACE_HEIGHT 21L
#undef  min
#define min(a , b) ((a) < (b) ? (a) : (b))
static char *Author   = "Perry S. Kivolowitz";
static char *ILibrary = "intuition.library";
static char *GLibrary = "graphics.library";
struct IntuitionBase *IntuitionBase;
struct GfxBase       *GfxBase;
struct Window        *w = NULL;
struct Screen        *s = NULL;
struct ViewPort      *vp = NULL;
struct RastPort      *rp = NULL;
static struct BitMap     fbm;
static int  sleepy_time = 4 * 12;

struct point {
long x;
long y;
long dx;
long dy;
};

struct Gadget close_gadget = {
NULL , 0 , 0 , FACE_WIDTH , FACE_HEIGHT , GADGHNONE |
GADGIMAGE ,GADGIMMEDIATE , BOOLGADGET , NULL , NULL , NULL ,
0 , NULL , HAPPY_FACE , NULL
};

struct NewScreen     ns = {
0 , 0 , 640 , 200 , 1 ,        0 , 0 , HIRES , CUSTOMSCREEN
, NULL , NULL , NULL , NULL
};

struct NewWindow     nw = {
0 , 0 , 640 , 200 , 0 , 0 , MOUSEBUTTONS | GADGETDOWN |
INACTIVEWINDOW ,NOCAREREFRESH | ACTIVATE | BORDERLESS |
BACKDROP ,&close_gadget , NULL , NULL , NULL , NULL , 0 ,0 ,
0 , 0 , CUSTOMSCREEN
};

/* produced by GI from a DPaint brush file */
unsigned short happy_face[63] = {   0x0000,0x0000,0x0000,
 0x000f,0xff80,0x0000,   0x00f8,0x00f8,0x0000,
 0x0387,0xff0e,0x0000,   0x0e7f,0xfff3,0x8000,
 0x19ff,0xfffc,0xc000,   0x37ff,0xffff,0x6000,
 0x2fe3,0xfe3f,0xa000,   0x6fdf,0xffdf,0xb000,
 0x5fbb,0xfeef,0xd000,   0x5fff,0xdfff,0xd000,
 0x5fff,0xdfff,0xd000,   0x6fff,0xefff,0xb000,
 0x2fbf,0x8fef,0xa000,   0x37df,0xffdf,0x6000,
 0x19e3,0xfe3c,0xc000,   0x0e7c,0x01f3,0x8000,
 0x0387,0xff0e,0x0000,   0x00f8,0x00f8,0x0000,
 0x000f,0xff80,0x0000,   0x0000,0x0000,0x0000
};

main(argc , argv)
char *argv[];
{
int result = 0;

if (argc > 1) sleepy_time = atoi(argv[1]) * 12;
if (sleepy_time <= 0) sleepy_time = 6;
IntuitionBase = (IBPtr) OpenLibrary(ILibrary , 0L);
GfxBase = (GBPtr) OpenLibrary(GLibrary , 0L);
init_happy_face();
if (IntuitionBase && GfxBase) {
while (result != HAPPY_FACE) {
wait_for_inactivity();
if (!(nw.Screen = s = OpenScreen(&ns)) ||
    !(w = OpenWindow(&nw))) goto out;
ShowTitle(s , 0L);
set_pointers();
set_color_map();
while (1) {
ScreenToFront(s);
if (result = zoom()) break;
ScreenToBack(s);
Delay(300L);
}
CloseWindow(w);
CloseScreen(s);
}
}
out:if (GfxBase) CloseLibrary(GfxBase);
if (IntuitionBase) CloseLibrary(IntuitionBase);
}

set_pointers()
{vp = &w->WScreen->ViewPort;
rp = &s->RastPort;
}
set_color_map()
{
SetRGB4(vp , 0L , 0L , 0L , 0L);
SetRGB4(vp , 1L , 0L , 0L , 0L);
SetRast(rp , 0L);
SetDrMd(rp , COMPLEMENT);
SetAPen(rp , 1L);
}

zoom()
{
unsigned long class;
register struct point *p;
struct point array[NPTS] , face_position;
register struct point *end_point = &array[NPTS];
long red , green , blue;
int style , loops;
struct IntuiMessage *message;

set_dots(array);
set_face(&face_position);
red = rand(9) + 5; green = rand(8) + 5;
blue = rand(8) + 5;
```

```
ramp_up(red , green , blue);
for (style = 0; style < 2; style++) {
loops = style == 0 ? 3500 : 2500;
while (--loops) {
if (message = GetMsg(w->UserPort)) {
class = message->Class;
ReplyMsg(message);
break;
}
p = array + (loops & 3);
if ((loops & 7) == 0) jerk_pointer(loops);
WaitBOVP(vp);
move_dots(0 , style , &face_position);
while (p < end_point) {
move_dots(1 , style , p);
p += 4;
}
}
if (message) break;
}
ramp_down(red , green , blue);
SetRast(rp , 0L);
if (message == NULL) return(0);
return(class == GADGETDOWN ? HAPPY_FACE : 1);
}

/*
* If you are not using MANX C or are using a version
* of MANX which does not have RangeRand (undocumented
* library call), simply replace this routine with one
* that returns a random number between 0 and X - 1.
*/

rand(X)
{
long seconds , microseconds;

CurrentTime(&seconds , &microseconds);
return((RangeRand((short) (1 << 15)) + 0xFFF & microseconds)
% X);
}

ramp_up(red , green , blue)
register long red , green , blue;
{
register long i;

for (i = 0; i < 16; i++) {
Delay(5L);
SetRGB4(vp , 1L , min(i,red) , min(i,green) ,
        min(i,blue));
}
}

ramp_down(red , green , blue)
register red , green , blue;
{
register long i;

for (i = 15; i >= 0; i--) {
Delay(5L);
SetRGB4(vp , 1L , min(i,red) , min(i,green) ,
        min(i,blue));
}
}

set_face(p)
register struct point *p;
{

close_gadget.LeftEdge = p->x = rand(580) + 20;
close_gadget.TopEdge  = p->y = rand(140) + 20;
p->dx = 2;
p->dy = 1;
BltBitMap(&fbm , 0L , 0L , w->RPort->BitMap ,   p->x,
          p->y , FACE_WIDTH , FACE_HEIGHT ,   0x60L ,
          0xFFL , NULL);
}

set_dots(array)
struct point array[];
{
register struct point *p;
register long i;
```

```
for (i = 0 , p = array; i < NPTS; i++ , p++) {
do {
p->x = rand(620) + 10;
p->y = rand(180) + 10;
} while (ReadPixel(rp , p->x , p->y) == 1);
while (!(p->dx = rand(13) - 6)) ;
while (!(p->dy = rand(13) - 6)) ;
WritePixel(rp , p->x , p->y);
}
}

move_dots(flag , style , p)
register int flag , style;
register struct point *p;
{
register long oldx , oldy;

oldx = p->x;
oldy = p->y;
p->x += p->dx;
p->y += p->dy;
if ((p->x >= (flag ? 640 : 640 - FACE_WIDTH)) ||
   (p->x < 0)) {
p->x -= p->dx;
p->dx = -p->dx;
}
if ((p->y >= (flag ? 200 : 200 - FACE_HEIGHT)) ||
   (p->y < 0)) {
p->y -= p->dy;
p->dy = -p->dy;
}
if (flag) {
if (style == 0) WritePixel(rp , oldx , oldy);
WritePixel(rp , p->x , p->y);
} else {
BltBitMap(&fbm , 0L , 0L , w->RPort->BitMap ,
   oldx , oldy , FACE_WIDTH , FACE_HEIGHT ,
   0x60L , 0xFFL , NULL);
BltBitMap(&fbm , 0L , 0L , w->RPort->BitMap ,
   p->x , p->y , FACE_WIDTH , FACE_HEIGHT ,
   0x60L , 0xFFL , NULL);
close_gadget.TopEdge = p->y;
close_gadget.LeftEdge= p->x;
}
}

jerk_pointer(counter)
int counter;
{
register long color_register = (counter % 3) + 17;
register long red , green , blue;

red = rand(10); green = rand(10); blue = rand(10);
SetRGB4(vp , color_register , red , green , blue);
}

init_happy_face()
{
InitBitMap(&fbm , 1L , FACE_WIDTH , FACE_HEIGHT);
fbm.Planes[0] = (PLANEPTR) happy_face;
}

long InputActivity()
{
register struct Task *p = FindTask("input.device");
register long return_value = 0;

Forbid();
if (p) {
return_value = (long) *(p->tc_SPReg - 2);
}
Permit();
return(return_value);
}

long TotalMem()
{
register unsigned long AvailMem();
register long availmem;

Forbid();
availmem = AvailMem(MEMF_FAST);
availmem += AvailMem(MEMF_CHIP);
Permit();
return(availmem);
}

wait_for_inactivity()

{
register long available_memory , last_available_memory;
register long repetitions = 0 , input_action ,
last_input_action;

last_available_memory = TotalMem();
last_input_action = InputActivity();
while (repetitions < sleepy_time) {
Delay(50L * 5L);
available_memory = TotalMem();
input_action = InputActivity();
if ((last_available_memory != available_memory) ||
   (last_input_action != input_action))
{last_available_memory = available_memory;
last_input_action = input_action;
repetitions = 0;
} else repetitions++;
}
}
```

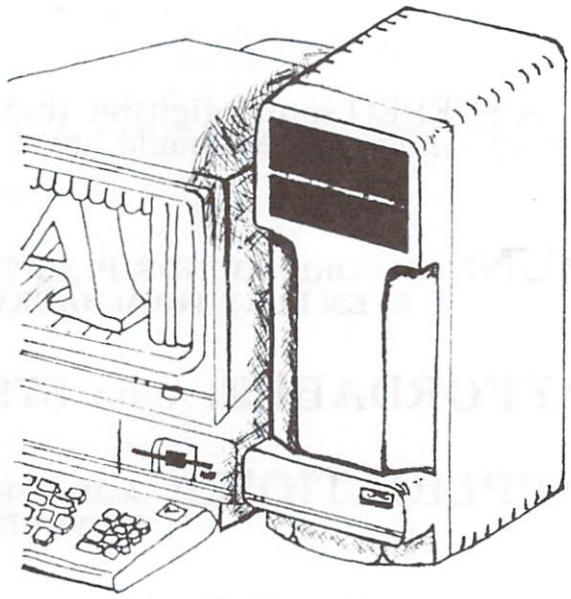
•AC•

Attention Amiga Owners

You should know the Side-Effects of owning your Amiga!!!

Side-ARM (Amiga Resource Module):

- 6/12 slot back plane
- optional 86 pin bus pass-through
- space for 2 half-height drives
- complete with standard power connectors



Side-ARM

Side-Store (memory card):

- 2 megabytes per card
- no wait states
- RAM-disk that survives resets/reboots

Side-Track (Disk & Clock):

- 20, 30, and up to 150 megabytes
- Reed-Soloman error correction code
- ST-506 compatible
- Battery backed real-time clock

Side-Band (Midi Interface, music synthesiser)

Side-Port (serial, parallel, SCSI)

Side-More.....

The following apply to all items:

- fully Zorro bus compatible
- auto-config standard
- matching plastic cover
- burned-in before shipping
- 6-month warranty

S
I
D
EFFECTS INC

6513 Johnsdale Rd.
Raleigh, NC 27615
Voice: (919) 876-1434
BBS: (919) 471-6436

Amazing Reviews...

LATTICE MAKE UTILITY

"...a good MAKE utility can become one of the most important components of your computing aresenal."

Reviewed by Scott P. Evernden
People Link SCOTT E
Compuserve 73116,3451

Sooner or later, you will find yourself working on a big project which requires many files. If you're a programmer, then you'll have the source and definition files for a program. A writer will deal with chapter, index, and table-of-contents files. Everyone has all the files they simply collect, print, catalog, backup and otherwise try to manage.

With so many files to keep track of, you'll likely find yourself forgetting their relationships, and the steps required to construct new files from the old ones. You make a modification here; another edit there. Before you know it you've lost track of what's changed and what hasn't. In desperation, you re-compile, re-format, or re-archive everything in sight since that's the only way you can be sure everything's up to date.

What you need, my friend, is a MAKE utility. What is a MAKE, you ask? Well, as its name implies, MAKE helps you to make things. It will help compile those programs that need compiling. It will generate new index entries for only those chapters most recently updated. And it'll backup and archive only those files which haven't been archived or backed up.

The original MAKE was born as part of the UNIX operating system during its formative years in the early 1970s at AT&T's Bell Labs. Stu Feldman created MAKE in order to maintain all the system software which together formed UNIX. To this day, MAKE is supplied as a part of the UNIX programming tools. It is used regularly by programmers, technical writers, system administrators, and just about anyone else using UNIX for any sizeable job.

MAKE learns about how to rebuild files which need rebuilding by following a set of instructions which you provide. The instructions are contained in an input file, read by MAKE, and otherwise known as a "makefile". MAKE reads the "makefile" to learn how to make things.

You'll need to understand a new language to talk to MAKE, but fear not because MAKE's makefile language is easy to learn. In its simplest form, you tell MAKE the name of a file (the "target") which needs making, followed by a colon, and then a list of files (the "dependencies") which are needed to make the "target". You say: this "target" is dependent upon these "dependencies". If I'm writing a User Guide for some new computer, its index will depend on all the chapters. The makefile for my project would contain a line:

```
book.index: chapt1 chapt2 chapt3
```

which tells MAKE that in order to make the target file "book.index" we need all the other dependency chapter files listed.

This isn't yet enough information, since we also need to tell MAKE how to go about actually constructing the index. We do this by adding lines after the one above. These "action" makefile lines are always indented with a TAB character, and provide MAKE with the actual commands that need to be performed or executed in order to construct the target file. For our index example, we might add the following line.

```
<TAB>genindex to book.index \
     from chapt1 chapt2 chapt3
```

Note that the backslash (continuation) character tells MAKE to continue reading on the next line in the file.

A complete makefile will probably contain many such target/dependency and associated action lines. We might add other rules like:

```
chapt1: ch1.sec1 ch1.sec2 ch1.sec3
  <TAB>join sec1 sec2 sec3 to chapt1
```

Most MAKE programs allow a "macro" facilty which allows you to provide a kind of shorthand for referring to files or actions. We could add a line near the beginning of our makefile:

```
COMBINE = join
```

and then rewrite the action given above as:

```
<TAB>$(COMBINE) ch1.sec1 ch1.sec2 \
  <TAB> ch1.sec3 as $@
```

A $ (dollar sign) indicates a use of a macro, and in this last makefile line, we've shown how to instantiate the COMBINE macro with parentheses preceded by the $. This line also demonstrates the use of a built-in macro (the $@). In this case,

this cryptic notation gets expanded into the target filename. MAKE programs provide a number of built-in default macros allowing replacement of certain dependency files, as well as parts of filenames.

One advantage to using macros is that they provide a facility for replacing definitions when the MAKE program is run. You can invoke MAKE like:

```
make COMBINE=myjoiner
```

and have a different program name run to actually construct the book chapters. Another example (for programmers) might have makefile lines looking like:

```
CC = lc
 CCFLAGS =
```

which defines macros indicating which particular C compiler to use in building programs within this makefile (Lattice C in this case), as well as any flags which we want to pass to the compiler. Here, we've defined none. Later, if we want to switch to Aztec C, we simply invoke MAKE as:

```
make CC=cc CCFLAGS=-l
```

and the different compiler and flags are used. Then, if the makefile has an action which looks like:

```
<TAB>$(CC) $(CCFLAGS) mypgm.c
```

we have complete capability to override the compiler and its flags.

Most MAKE programs provide further capabilities to allow you to state in a concise way a generic set of actions to perform in order to "transform" one kind of file into another kind of file. For example, we could add lines to our programmer's makefile:

```
.c.o:
 <TAB>$(CC) $(CCFLAGS) $*
```

This is a generic rule stating how to change (or transform) a file ending whose filename ends with a ".c" into a new file ending with a ".o". Later in the makefile, we need only indicate the target and its dependencies, and, if no actions are provided, MAKE will use this generic action.

The "$*" default macro shown is another shorthand offered by MAKE, and will have the prefix of the dependency filename (the filename up to the extension) substituted for it whenever this action is invoked.

So, we could state:

```
mysub.o: mysub.c mysub.h
```

meaning that the object file for mysub needs rebuilding if either the C source file, or the header files change. If no indented action lines followed, MAKE would refer to the generic transformation rule given above and then do the following:

```
lc mysub
```

If we had instead invoked MAKE like this:

```
make CC=cc CCFLAGS=-l
```

then MAKE would execute the following command:

```
cc -l mysub
```

There is much more that could be written here about MAKE and how to use this useful tool. We've only examined the basics here. Suffice it to say that, due to its power and flexibility, a good MAKE utility can quickly become one of the most important components of your computing arsenal.

What's TOUCH?

A TOUCH program is a useful adjunct to a MAKE program because it provides an easy way to change the date of last modification of a file. You simply say:

```
TOUCH filename
```

and the file's creation date will be set to the current time. This is sometimes useful if, for example, you simply want to record when MAKE last did something. One use might allow printing of only changed files. The makefile would have lines looking like:

```
CH1_FILES = ch1.sec1 ch1.sec2 ch1.sec3
CH2_FILES = ch2.sec1 ch2.sec2
PRINTIT = copy to prt: from

print: $(CH1_FILES) $(CH2_FILES)
<TAB>$(PRINTIT) $?
<TAB>touch print
```

This short example illustrates how to tell MAKE to print only those files which have changed since they were last printed. This is done by keeping around a dummy or flag file, here named "print", whose sole purpose is to maintain its own creation date.

When MAKE tries to rebuild the target "print" file, it will take action only on those files which are younger than the file named "print". This is specified by yet another default macro (the $?) which gets replaced by a list of all the dependency files which are newer than the target file.

The TOUCH action is the key one here, since it then causes the file named "print" to have an updated timestamp, set to the current system time. As a result, MAKE will no longer regard the freshly printed files as being newer than file "print", until any of those files gets modified.

Lattice LMK and TOUCH

Lattice supplies both its MAKE, named LMK, and a TOUCH program on a diskette along with several example makefiles taken from its documentation. A hardcover spiral bound book consisting of less than 50 pages of instructions complete the LMK package.

LMK is a moderately capable MAKE program sporting many of the most desired features of a full-blown MAKE utility. It lacks some of the more esoteric functionality usually found in the UNIX versions. In the course of using LMK and TOUCH for the past several weeks for my own work, I have noted a number of deficiencies and problems which make LMK a less than perfect tool.

The first thing I saw when reading the LMK manual was that one of the example makefiles listed included a flag to the compiler (the LC Lattice compiler, of course) which I recognized as being valid only for the MS-DOS Lattice compiler, and not Lattice/Amiga C.

The TOUCH program is not documented at all in the manual; you discover later that the TOUCH instructions are in a short file on the diskette.

In general, however, the thin manual is acceptably well written, and leads the reader from simpler to more complicated MAKE concepts. An 8 page Reference Manual is included in the instruction book as a concise synopsis of how to use LMK.

Upon examining the LMK diskette for the first time, I noted that it was only 12% full. I don't know about you, but for the $125 cost of LMK, I expected a disk FULL of programs. Perhaps many useful makefiles could have been included for different kinds of work and compilers. A script file "loadlmk" is included to copy LMK and TOUCH to your C: (commands) directory.

Problem is, these program files are BIG! TOUCH is about 19K and LMK tips the scales at over 48K. I don't have 67K (133 blocks) of spare space on ANY of my SYS: disks, and particularly not the one I use for C programming; maybe you do.

For me, LMK and TOUCH joined the other programs I must keep on every working disk I use. The large program size of LMK is of concern. Since LMK remains in memory while it does its work, a fairly sizeable chunk of precious memory is consumed. This memory is therefore not available to do other things.

After making a working copy of the disk, I tried LMK on the included example demo program. On the evaluation diskette I received, this demo proved to be an extremely uninspired program which simply printed out some lines of text.

You will soon discover, as I did, that LMK is plagued by a problem I have witnessed in other programs compiled with Lattice/Amiga C; it sometimes doesn't do anything. The program loads, prints an initial copyright message, and then exits. The program thinks you have typed a CONTROL-C to stop processing.
When this happens (and it happens a lot!) you have to run LMK a second time to get it to "take hold". While using LMK over the last weeks, this one problem alone has caused me much aggravation, since I must actually sit in front of my Amiga to make sure that LMK actually starts running. If I didn't, then I might be lead to believe that LMK decided everything was up-to-date and did no work.

Beyond this problem, LMK is also frustrating in its use of transformation rules (then ".c.o" target example above). You must explicitly indicate the target in the makefile to get the transformation action to run.

In a more full-featured MAKE utility, the program is usually smart enough to look in the current directory for dependencies needed to build the targets of transformations.

In other words, if a ".o" file is needed (for an ALINK step, for example), then a good MAKE program will automatically look for the corresponding ".c" file in the current directory, and then perform the transformation actions. LMK needs more explicit directions.

Due to problems in AmigaDOS, LMK is unable to detect when an action it has invoked has failed for some reason (a missing file; a compile error). This ability is critically important to getting the best use of a MAKE program, since you generally want MAKE to stop running upon encountering errors.

Instead, LMK forges onward performing each of the rules in makefile, constructing target files based upon possibly out-of-date files. I always had to pay close attention to make sure that if errors occured, I could CONTROL-C out of LMK. It is possible to avoid this problem, by adding rules to explicitly delete target files before thay are remade, but this forms a poor solution.

Another problem occurs whenever you try to specify a target whose filename includes a colon. For example, an LMK makefile with a target/dependency line like:

```
df1:object/output.o : output.c header.h
```

always caused LMK to complain of makefile syntax errors since, due to the ':' (colon) character, it sees this line as having something named "df1" as a target.

This is a generic problem in the syntax of the makefile language when combined with AmigaDOS device naming conventions, and needs specific treatment in the AmigaDOS environment. I could neither locate instructions in the manual, or fabricate solutions to get around this problem. I repeatedly wished Lattice had accounted for this problem in some way.

LMK appears to slowly consume memory as it is used. I suspect that a known bug in the the current implementation of the AmigaDOS Execute() function is the culprit here, and not Lattice.

The TOUCH program supplied as part of the LMK package exhibits more problems. I tried using TOUCH in my makefiles to update the timestamp of a file named "Now" on the SYS: disk. This, I thought, would ensure that the system time after any re-boots would become updated when the SYS: disk is validated.

It never worked; my system time at next re-boot was always invalid. DATE would responded with "<invalid> <invalid> <invalid>". Furthermore, the file "SYS:Now" would NOT appear when I LISTed the SYS: directory, but it did show up in a DIR listing. Peculiar?

TOUCH also has a problem in locating files with certain filenames. I had a file named "rayhit.c" which TOUCH simply refused to believe existed. Repeated attempts to:

```
touch rayhit.c
```

always yielded:

```
Can't find file(s): rayhit.c
```

even though file "rayhit.c" plainly appeared when LISTed. Other invisible filenames are easy to trip over; files named "scott.ev" and "buggy-touch", for example, also can't be touched with this TOUCH. There is clearly some sort of problem in the way Lattice TOUCH traverses directory structures.

Conclusions

Despite the many annoying problems I had with LMK and TOUCH, I did find that, with care, they could be usefully applied in many day-to-day activities on my Amiga.

LMK does support all of the most common MAKE capabilities including comments, the .DEFAULT rule, grouped dependencies, echo suppression, command line macro substitution, and the unconditional (-u), silent (-s), and no-build (-n) options. The ignore-errors flag (-i) is provided, but it doesn't work, as explained above. A nice idea is the help (-h) option which lists all the other LMK command line option as a handy reference.

LMK does seem to do a good job of recognizing circular dependencies (file "A" depends on file "B" which depends on file "A"). As with the UNIX versions of MAKE, LMK also provides the debug (-d) option, which produces an immense amount of debugging output on LMK's progress which I personally find quite useless.

A facility called "local input files" is unique to LMK, and provides a mechanism for supplying input files to a program as specially quoted inline text in the makefile. LMK can be run from the Workbench (an icon file "lmk.info" is provided), however, I can't quite figure the usefulness of such a capability (perhaps as an archive/backup facility?). I do question the rather high price tag on this package. At a list price of $125, LMK comes close to the price of the Lattice/Amiga C compiler at $150, but does not, in my estimation, provide a "bang-for-the-buck" ratio I feel comfortable with.

If you should decide that you need a MAKE program, then you would be well advised to do some careful shopping. You should make sure that some of the problems which I have observed using Lattice's LMK either don't exist, or won't bother you.

Lattice Make Utility, version 1.00B, $125

Lattice, Inc.
P.O. Box 3072
22W600 Butterfield Road
Glen Ellyn, IL 60137

•AC•

The AMICUS Network

New AMICUS & Fred Fish Public Domain Disks and more...

By John Foust

Sometimes the universe plays tricks on you.

Telecommunications is a frequent topic of this column. There is a reason for this. I spend a lot of time online to networks. Most of the programs in the AMICUS collection pass through my modem. All of my Amazing Computing work is sent via modem to PiM Publications, Inc.

While preparing the last issue, I sent several stories to PiM through a different service than I usually do. My normal method of sending stories failed; each message was unreadable by the addressee, and the network was acting strangely.

On another computer service, I sent the stories to the ID I thought was PiM Publications, Inc. When it became apparent they never arrived, I double checked the ID. The ID was wrong. This is the equivalent of having a wrong address on an envelope. The computer, stupid as it is, delivered the stories to the wrong person without question. I wrote a letter of apology to this anonymous ID, and asked them to delete the files.

A week or so later, I got a reply from this user. Chance would dictate that this 'wrong number' was an unused user ID, or someone who would take no interest in these yet unpublished Amazing Computing articles. Instead, this ID belonged to the Amiga editor of another Commodore magazine. He laughed at the mix-up. We exchanged gossip and a copy of our respective magazines.

Disk speedup

If you ordered an AMICUS disk recently, perhaps you noticed that the icons pop up a lot faster that you'd expect. The disks were recopied under Workbench version 1.2. CommodoreAmiga has improved the algorithm for storing files on disk, and it is backward compatible with 1.1. If you have a beta copy of Workbench 1.2, you might want to format a new disk, and then do a 'copy df0: df1: all' command in the CLI, from the old disk to the blank disk. It takes a while, but the results are worth the time and effort.

AMICUS #11 bug fix

Last month, words were omitted from my description of the picture conversion utilities on this disk. There are four programs here that read Commodore 64 picture files. They can translate Koala Pad, Doodle, Print Shop and News Room graphics to IFF format. Of course, getting the files from your C-64 to your Amiga is the hard part.

AMICUS #12

In the first AMICUS column, I said we might expect some high-powered public domain software, including compilers and assemblers. I forgot to mention linkers, the program that glues together the output of an assembler or compiler.

A group of Amiga programmers called the Software Distillery have produced a public domain linker, fully compatible with the standard 'Alink' linker from MetaComco. They call it 'Blink', and it can be found on disk 12. It is about 25 percent faster than 'Alink'. Documentation is included. Future versions will include overlay support, and they think they can make it even faster. They also included the 'faster' option supported by 'alink', just for compatibility's sake.

Another super program from the Distillery is PopCLI. In the startup sequence file on your Workbench, 'RUN' the PopCLI program. Then, like the popular SideKick program, press Amiga-ESC at any time, and a new CLI window will appear on the screen. PopCLI also has a screen-blanking feature that will change the screen color to black after a given time of non-activity at the keyboard, to save the phosphors in your monitor. To restore the screen, press any key, such as shift. The C and assembler source code is included for PopCLI.

MenuEd is a C programmer's development tool. It lets you create a series of menus for a program, using a simple menu-driven interface. You can add items to each menu, and then save the whole design as a series of C structure declarations, ready to compile into your program. This shareware program comes from David Pehrson at Mantis Development.

The other C source code here includes an example of using VSprites, the Amiga virtual sprites, and 'spin3d', an animated demo with dozens of rotating cubes.

The executable programs include 'clean', a disk cleaning program. It spins the disk for a given number of seconds, so you can use a disk cleaning disk according to the instructions.

The 'epsonset' program presents a screenful of all the text modes available on an Epson printer. Click on 'condensed',

and the proper escape code is sent to the printer, through the 'PAR:' port.

The 'showbig' program will display a high resolution picture as a low resolution screen. Insert a joystick in the back mouse port, and you can scroll over the entire picture.

'speaktime' will speak the current time, either from a click on the Workbench icon, or from the CLI.

'undelete' is a CLI command to undelete a file. However, you must be able to remember the former file's name to use this.

Three picture converting programs will translate Apple][pictures to IFF format. The three will handle low, medium and high resolution Apple pictures.

Dave Devenport, of Copperstate Business Systems in Arizona, wrote two disk copying programs that work faster than 'diskcopy'. One, called 'quick1', copies a disk in record time, but 'quickEA', the second copier, will copy the latest Electronic Arts games, and remove the copy protection forever.

The documents on AMICUS 12 include a list of Amiga software and hardware vendors, compiled by Amazing Computing music editor Richard Rae.

Also present is instructions for fixing the early Cardco memory expansions. The first 150 boards have an extra wait state in the memory, which means they run slightly slower than they should. This text explains which traces to cut and jumpers to set to remove the wait state.

A cross-reference to the Amiga C 'include' files shows which 'include' files include other 'include' files, in chart form.

Another text explains the format of the scripts for the SlideShow program that produced the demo on the Electronic Arts Kaleidoscope disk, supplied with every Amiga.

This disk has a list of tips for playing the game 'MindWalker'.

Three beautiful high resolution pictures are included, along with the SeeILBM program. These can be displayed from the Workbench, at the click of an icon, without a paint program. One is a three-dimensional Mandelbrot picture, another is a robotic arm grasping a cylinder, and the third is a picture of a star-destroyer class starship from the Star Wars movies.

This disk has Ewan Grantham's Amiga bulletin board system, written in Amiga Basic, as I promised in a column long ago.

A demo of TxED 1.3, the text editor, is on AMICUS 12. This demo does not load or print, but it shows a tightly coded user interface. This program is written by Charlie Heath.

The 'star10' program makes moving star fields, similar to those in the opening sequence of Star Trek. The assembly language source is included.

AMICUS 13
This disk has several examples of using libraries from Amiga Basic.

The first is Carolyn Scheppner's library examples for displaying IFF pictures. Scheppner is a member of the technical support team at Commodore West Chester. Her method uses a new IFF picture type more suited to Basic's style of displaying graphics. A program converts standard IFF pictures to this format, and another example displays the new picture. An example of a screen printer from Amiga Basic rounds out the Scheppner examples.

Also present is my FutureSound library example to interface IFF sounds to Amiga Basic. FutureSound is a sound digitizer from Applied Visions. I do not have any stake in the company. However, I wrote the library for Applied Visions after I read a plea from Dan Lovy on Compuserve. Lovy is an employee of Applied Visions. Their shipping deadline was rapidly approaching, and they needed this software interface done quickly. At that time, I was researching the technique, as a hopeful topic for a future column.

The assembly language source to a library is included, based on an example from Commodore-Amiga. The C source to the functions to load and play IFF sounds is here, too. The library code shows the proper interface from assembler to C, as discussed in Gerry Hull's C tutorial from last month.

For astronomy enthusiasts, there is a version of the 'gravity' program from Scientific American, January 1986. This program simulates the motion of two bodies revolving around a third, larger body. It draws this relationship on the screen, in color. Several example data sets are provided.

AMICUS 13 has complete instructions for making your own MIDI interface. For more information about MIDI, read Richard Rae's Amiga Notes columns here in Amazing Computing. 'MIDI' stands for 'Musical Instrument Digital Interface'. It is the present standard for interconnecting synthesizers and computers.

I must apologize for announcing AMICUS 11, 12 and 13. I had not finished preparing them when I wrote my column two months ago, and many orders for AMICUS disks were held up, until I sent the master disks to Don Hicks. If your order took a long time, blame me, not PiM Publications. I can say I will have several more AMICUS disks to describe in the next column. I will say nothing more, to keep myself out of trouble.

User Group Newsletters
I appreciate the user group newsletters that some user groups have sent. A user group newsletter is a lot of work. Some of them look more like magazines, in terms of layout and article content.

The names of newsletters and user groups demonstrate the 'creative edge' typical of Amiga users: the Amiga Guru, from the CA-AUG group in Ohio, has a column called the Bit Bucket; there is a Space Coast Amiga User Group in Florida; and a newsletter called Gadget from the Causer group in South Carolina.

I have been in touch with Georgio Cupertino, an Amiga developer in Monaco. He produces the Cupertino Newsletter for developers in France and Monaco. It is written in "franglish", a blend of French and English.

Beyond France and Monaco, the magazine has heard from Amiga users in New Zealand, West Germany, Puerto Rico and Japan.

New Fred Fish Disks
Close to press time, I found the listing of the contents of the newest Fred Fish disks, numbers 31 through 35. I don't have them in hand, but I can describe some of the programs on the new disks.

Fred Fish Disk 31
This disk has a version of the game of 'Life', not the plastic car and kiddies version, but the cellular automata simulation. This version is very fast, since it uses the smarts of the Amiga blitter chip to calculate the next generation of the cells in the universe. It can calculate over 19.8 generations a second, in a 318 x 188 universe.

This disk has version 3.0 of the Robert French and RJ Mical Mandelbrot program. I know, yet another Mandelbrot program.

There is a replacement for the Manx 'set' command, with improvements. This program lets you set 'environment variables', a method of setting the default parameters of cooperative CLI-based programs.

'Tree' draws a recursive tree, not the directory-tree type, but the green type. This disk has the crippled demo of 'TxEd', described above. 'Xicon' is a program to execute AmigaDOS scripts from an icon, and 'Ticon' will view a text file from an icon. There is an example of a mutual-exclusion gadget, and a program to measure the relative performance of CHIP and FAST memory.

'VDraw' is a full-featured drawing program by Stephen Vermeulen, based on the original FreeDraw program from the early AMICUS disks. 'FreeDraw' was written by Rick Ross, as a learning toy, before he wrote Marauder, Grabbit, and Exactly!, at Discovery Software.

Fred Fish Disk 32
This disk has three Amiga Basic programs, an extensive address book, and a calendar program, and the Sword of Fallen Angel, a text adventure. Another David Addison ABasic game, Solitaire, is here.

This disk has the Dos Plus series of shareware CLI utilities from Bill Beogelein. It comes in two volumes. The first has a simple calculator which adds, subtracts, multiplies and divides pairs of numbers, utilities for converting between ASCII characters and their value, a binary, decimal octal and hex converter program, a 'wc' type program to count the number of words in a text file, and a CLI help program.

Volume 2 has 'col', a program to set 60 or 80 column screens, without Preferences, a program to reset Preferences to the last saved settings, or the default Preferences; 'lock' and 'unlock', to set the protection bits on a file, to prevent it from being deleted, a program to strip binary from files, so they can be read with the 'ed' editor; a timer that counts time in minutes and dollars, for use on pay networks or long-distance phone calls, and a printer configuration tool to set-up any printer. Also included is a simple sliding square puzzle game.

This disk has the 'MacView' program described below. This disk doesn't have the example Mac pictures, only the executable. There is a 'ShowHam' program to view HAM mode pictures from the CLI.

The 'spin3' and 'trails' programs are here. These are the same programs as on AMICUS 11 and 12, described last month.

Fred Fish Disk 33
'Bigmap' is a low-level graphics example, which scrolls a large bitmap with the ScrollVPort() function. There is another C example of double buffered animation, using BOBs and VSprites.

'DiskMapper' is a program to view which sectors (blocks) on a disk are currently filled with data. You might use this to aid in recovering trashed disks, or for use with the Tracker program, to learn which sectors of a disk are used.

Leo Schwab, who also goes by the anagram handle 'Bols Ewhac', wrote two very interesting programs on this disk. He wrote Oing!', an extension of the Boing! program. This version has nine or so small boing-style checkered balls that bounce in front of your CLI or Workbench screen.

He also wrote 'memview', a program to view the memory of your Amiga, moving through memory under control of a joystick in the rear mouse port.

'Sproing' is another version of this program, with sound added, so nine balls are bouncing, and each boings as it hits a screen edge.

(AMICUS 14 will have these programs, plus 'Zoing', an extension of Oing! that obeys real-world physical laws of conservation of momentum. One ball is under control of the mouse, and you can use it to knock the other balls around. It is an intellectual air-hockey game, if you will.)

'ScreenDump' dumps the highest screen or window to the printer, in C, with source code provided. 'Sdb' is a simple database program from a DECUS tape. 'stars' is a star field graphics demo, like the opening sequence of Star Trek. Also included is a 3D version of this same program, for those of you with an extra set of 3D colored glasses.

'TermPlus' is yet another terminal program with file capture and send, a dialing library, programmable function keys, Xmodem, and the Compuserve-B file transfer protocols. 'Vt100', version 2.0 by Dave Wecker, is a very close emulation of the standard VT-100 terminal. It has Kermit and Xmodem file protocols, and scripts for automatic logons, and programmable function keys.

Fred Fish Disk 34
This disk has files for using Gimpel's Alint C program syntax checker, and 'blink', described above. 'browser' is an improved version of the program on Fish disk 18, converted to Manx C.

This disk has 28 new fonts for the Amiga, from Bill Fischer, in several sizes and styles, most larger and more informal than the standard Amiga set.

'Pr' is a background print utility, with options for setting printer style. You can use wildcards to specify which files to print. There are two versions of generic b-tree database examples here.

'Calendar' is an appointment calendar, with alarm. 'less' is a pun on the Unix 'more' program, a program to view a text file. You can search the file for given text, or position within the file, at a given percent, or by line number.

There is an example file requester, in the style of the Deluxe Paint requester. It comes with a source code example.

Fred Fish Disk 35
This disk has several C examples written by Commodore-Amiga employees. One demonstrates making asynchronous I/O calls to a DOS handler; another shows the way to get an Intuition pointer to a CON: or RAW: window under 1.2; another is a dual playfield example, which shows a 400 x 300 x 2 bit plane playfield on a 320 x 200 x 2 plane deep playfield; another is a set of general purpose subroutines for sending AmigaDOS packets.

It has two versions of DirUtil. This program presents an Intuition gadget-oriented view of the files on a disk. You can delete and rename files, change subdirectories, all with the mouse, and very little typing.

It has an object file version of Charlie Heath's well-known file requester, for Lattice C. Heath's file requesters in the TxED editor are fast and easy to use. They don't make you wait to load a file from a requester. The filenames are shown as soon as the requester appears, and you can click on one immediately, and proceed with the program, without waiting.

There is a program to view Macintosh pictures in low or high resolution, by Scott Everndon. With a screen-saving program such as SaveILBM, you can transfer Mac pictures to IFF format.

'Plop' is a simple IFF reader program. 'Tsize' will print the total size of all files in a given subdirectory. This Fish disk has 'PopCLI', described above, plus three Devenport disk copiers, including one that isn't on the AMICUS disks. Sorry, I must have missed it.

It has 'SpriteMaker', Ray Larson's shareware sprite editor. This can save the sprite as C data structures, for use in C programs.

'Tracker', by Brad "Lord Bradford" Wilson, will read a given set of tracks from a disk, and convert the data to a single file. In this way, a disk can be broken into several files, ARCed to smaller sizes, and transmitted via modem.

On the receiving end, the files can be re-constructed to an exact copy of all the files and data on the disk, including file names, date, and subdirectory structure. Since it reads absolute tracks, even a Kickstart disk can be sent via modem. Some people were using this to send version 1.2 of the operating system to their friends across town.

This disk has a tremendous shareware game called 'TriClops'. It was to be a commercial product, but the authors, Geodesic Productions, decided to give version 1 away, and sell version 2.

Simply put, it is a 3-D space invasion game. It uses fractal geometry to create planet surfaces and surface objects. Future versions of the program will allow user editing and saving of the planet characteristics, such as building designs, and even let players exchange objects, using a standard database format. It will have a two-player mode, where you can play via modem.

It is semi-real-time graphics, with filled polygons. The colors are cartoonish, but it looks like a lot of fun.

The obscure programs on this disk include 'Unifdef', a C preprocessor to remove given #ifdef'd sections of a file, leaving the rest alone, and a VT-100 emulation test program. This requires a Unix system, and it verifies the VT-100 compatibility of a terminal or terminal emulator.

Developer Conference

The second Amiga Developer Conference was cancelled, see the Roomers column for more information. I had hoped to present the loot from the conference in my next column, but it will have to wait until the December or January issues.

In the next column, I might have goodies from the West Coast Commodore Association show in Los Angeles, either programs or articles and pictures.

I hope to interview more Amiga people at the show, in the style of the RJ Mical interview in a past issue of Amazing Computing. Of course, the developer conference should present many interview opportunities, too.

This might include third-party Amiga developers, instead of Commodore-Amiga employees, past or present. I enjoy interviews for several reasons, even though the transcription from tape is such a tedious process. An interview gives insight into the thought processes of the people who helped shape the Amiga.

•AC•

A Tale of Three EMACS

... an editor is expected to fit one's style as a carpenter's favorite hammer fits his hand.

By Steve Poling
People Link SPOLING

Why bother with Emacs? What is Emacs and where did it come from? I will answer these questions in this article and then proceed to compare three public domain versions of Emacs-like editors available for the Amiga.

Many of us spend a lot of time editing files. As a result, an editor is expected to fit one's style as a carpenter's favorite hammer fits his hand.

Unfortunately, ED and EDIT, the editors provided with the Amiga, are not all one wants in an editor. Each of the EMACS-like editors reviewed here are better than ED and EDIT. I recommend replacing the ED and EDIT with one of the public domain editors described here.

Where did EMACS come from? Richard Stallman at MIT wrote the first EMACS. His editor has been widely copied since then.

Gossling at Carnegy Mellon University wrote another version of EMACS that runs under Unix, and has been adapted to run on other machines. That version of EMACS is sold by UniPress software.

A few years back, an effort arose to write a subset of EMACS for the IBM PC. It was called MicroEMACS, and placed into the public domain. MicroEMACS is the direct ancestor of all the EMACS-like editors I will describe here.

Another development that warrants watching is the Free Software Foundation and its GNU project. The man who started all this stuff about EMACS is Richard Stallman who also has started the Free Software Foundation. He is working on a public domain version of Unix, called GNU. An outgrowth of this effort is GNU-EMACS. This is a full-featured EMACS with a Mock LISP interpreter embedded within it. I hope that someone is working on a port of GNU-EMACS to the Amiga right now. If you know anyone who is, please help me get in touch with them.

Until GNU-EMACS is available, there are other decendants of Micro-EMACS available. So many in fact, that I spent some time going thru three easily available implementations comparing them to decide which one I wanted to use. I thought you might benefit from my experience.

I looked at the two versions of EMACS on Fred Fish Disk number 22 and the version of EMACS found on Fish Disk number 23.

(Emacs purists are now gritting their teeth. When I refer to Amiga versions of EMACS, you should all recognize that I mean versions of Micro-EMACS. Please excuse this literary licentiousness.)

I will refer to these 3 implementations of Micro-EMACS for the Amiga by the names LEMACS, PEMACS and GEMACS. If you pick up the Fish Disks in question, you'll have no problem identifying which version I mean.

LEMACS was my least favorite implementation of EMACS for the Amiga. Yet, in one important sense, it is the most advanced version of EMACS for the Amiga.

It supports EMACS modes of editing. When an editor knows enough of the syntax of a programming language, it can help you by completing syntactic constructs automagically. It saves you the frustration of having to wait for the compiler to catch your syntax errors. It helps you generate syntactically correct code in the first place. This feature is the real joy of a full featured EMACS implementation.

To some extent, this is supported in LEMACS. And this is a major plus for this version. For example, it knows you might be editing a C program, and it will type a close-parenthesis ')' after one types an open parenthesis '('.

Since I was a new user of LEMACS, I appreciated the implementation of the HELP key. When the HELP key was pressed, a help file was loaded off disk, put into a buffer, and displayed on the screen. This was most appreciated, since I was used to a different set of key-bindings from another EMACS and was quite a problem trying to get LEMACS to quit without saving.

On the down side, LEMACS didn't support the Amiga keyboard as I would like. I could position my cursor using the arrow keys, which is a plus. But it did not provide key bindings for the function keys on the Amiga keyboard.

It did not take advantage of the Amiga mouse. Moreover, it did not use any of the Intuition Menu interface. I've become spoiled by the mouse when editing, so I marked down LEMACS for this reason.

I would recommend this version if you aren't spoiled by the mouse and the menus yet and intend to use this editor to do a lot of program source code editing. Otherwise, read on.

I said that LEMACS was the most advanced version of Micro-EMACS for the Amiga. On the other hand, PEMACS is the version best customized for Amiga.

It supports all the Amiga function keys, it supports the Arrow keys. It lets you enter META-key commands via the ALT-key sequence.

Emacs commands are issued via pressing control key sequences. For example, you type CTRL-V to move down one screenful of information. To move up one screenful of info, you type the ESCape key followed by the V key. This is known as META-V in the EMACS documentation jargon.

I prefer the ALT-key sequence to entering ESC-key, because it is a closer analogy to hitting CONTROL-key.

If you find your cursor at one end of the screen, and the text you want to change at another point, you merely click your little mousie at that point and your cursor is THERE. I got spoiled by this feature soon after I learned of it.

I must admit that when I studied PEMACS to make this report, I was familiar with it. Thus, it was very easy to grow to rely upon the menu selection bar atop the window instead of memorizing all the key sequence commands for obscure editing functions.

Menu support is something you get spoiled by rather quickly as well. But let me lobby here for giving the user BOTH menus and obscure key bound command sequences. We're not on the Macintosh, you know.

The worst misfeature of this implementation is the binding of the help key to CTRL-U. This key is right next to the backspace key and the up-arrow key. It gets hit by mistake real easy. Now, CTRL-U is a very useful key. It is bound to the repeat-command function. So, if you hit CTRL-U, EMACS will repeat the next command you enter some number of times. (It defaults to four times.)

If you hit HELP and then inadvertantely hit DELete you can kiss 4 characters good bye. Or if you hit HELP then CTRL-V, you are 4 pages away from where you started, lost and wondering what happened.

I recommend that if you are not using LEMACS, take a look at PEMACS to appreciate what a boon editing with a mouse is. However, I'll neglect to recommend this version over GEMACS which I'll describe next. The only negative about this implementation is that it lacks the editing modes found in LEMACS.

I recommend GEMACS as the Amiga version of Micro-EMACS. As the documentation for GEMACS states, "Accept No substitutes." Are you confused by the plethora of different public domain EMACS-like editors? I am too.

GEMACS is a step backwards. It is designed to be more compatible with GNU EMACS. GEMACS is the version you SHOULD USE. The proliferation of MicroEMACS implementations is confusing.

All these implementations should be brought together into a common version which supports the best of each of the individual EMACS implementations which are available. At present, there is an effort on USENET to make GEMACS a jumping-off point for a single common implementation of MicroEMACS.

GEMACS comes on Fish Disk 23 in a most confusing form. There are four executable versions of GEMACS available on Fish Disk 23. Two of the versions provided are plain-jane implementations without support for the mouse or the menus.

I think they are provided for the EMACS purists who are in a snit because they aren't using a LISP machine. These two stripped down GEMACS implementations are produced using the Aztec and the Lattice compilers respectively.

There are two more executables which support all the function keys, the arrow keys, the mouse and provide menus atop the editing window. As noted before, there is an executable provided which is generated by the Lattice and the Aztec compilers. (Green Hills must feel left out.)

I did not look very closely at the two plain-jane executables. I was surprised to find that the Lattice generated executable would not run properly on my machine. I'd be interested to know why. However, the Aztec generated executable is about half the size and it seems to run well. This is the executable I'm writing this article with.

GEMACS supports the most versatile menu structure of all the implementations reviewed. The HELP key is implemented differently from both LEMACS and PEMACS. When you type HELP, the editor waits for you to type another command key sequence. When you type that command key sequence, it responds with a one line description of that function.

On the down side, you have to learn not to use the BACKSPACE key. That key is bound to the help-command as well. I suppose it keeps you from inadvertantly zapping characters when you request help.

Forget what shifted-F5 does? Simple, just type HELP followed by shifted-F5. That's easy, shifted-F5 is bound to the command "delete-other-windows".

As is the case with PEMACS, GEMACS does not support editing modes. I hope the authors of LEMACS conspire to add editing modes to GEMACS. Of course, should GNU EMACS ever become available on the Amiga, I'll drop everything and go to that. In the mean time, I intend to use GEMACS and recommend it to others.

•AC•

.bmap File Reader in AmigaBASIC

I wonder what's in those .bmap files... Hmmm...

By Tim Jones
People Link ABSOFT 1
SYSOP The Window BBS (617) 868-1430

While programming in AmigaBASIC, I was confronted with a challenge. I was attempting to work on a terminal program that would be able to call up a 'list' or allow you to 'Spawn' a new CLI from which you could run DOS commands.

While looking through the various demos that are included on the Extras disk, I noticed a file that included a routine (via a call to a .library) that used the DOS 'EXECUTE' command. I was sure that there in would lie the answer to my challenge.

After probing the inner workings of the program, I realized that, given the appropriate .bmap files, I would be able to acces most, if not all, of the Amiga's built in functions. The .bmap file that was called by this program was the dos.bmap file (as "dos.library").

After looking at the way the program seemed to interact with the library, I was quickly stumped and turned to the explanation of .bmap files in the manual. This in turn brought up more questions; mainly, if I could get into the .bmap file, would I be able to make use of what was there?

I used the FileZap file that's been floating around the area BBS's and found that there was a lot more in the dos.bmap file than just 'EXECUTE". I started prowling around my developer's manuals and found that the library calls represented in the .bmap file were straight out of the AmigaDOS manual.

After comparing the data that I saw using FileZap with the format that the BASIC manual said should be there, I became totally confused. With the exception of the routine names, the file appeared to contain useless garbage that was supposed to represent the register usage of the routine.

As with most of the programs that I write, I decided that there must be a more orderly way to examine the file and determine exactly what was available. Thus 'BmapReader' came into existence as a way to examine the .bmap files that were appearing on various BBS's across the nation.

Programming Notes

When you load and run 'BmapReader', it will look to the default directory for the graphics.bmap file. It will open the library and use two of the routines that are available. These are the Move() and the SetDrMd() routines.

The Move() routine allows you to specify a pixel address, rather than a character cell, when positioning text on the screen. The call is invoked with the following syntax:

CALL Move&(rastPort&,xLoc,yLoc)

You can get the value for the rastPort pointer from the WINDOW(8) function. In all of my programming, I have made it a habit to set the pointer using the variable name Rp&.

This is done by calling the WINDOW(8) function AFTER I have set up the window structure. In the event that I have more than one window to work with, I will set them up according to the window numbers (i.e. WINDOW 1 = Rp1&, WINDOW 2 = Rp2&, etc.).

One important fact to keep in mind when using Move() is the rastPort address is the upper left hand corner of the window and does not take into account the borders or the title/drag bar. Therefore, when you position things on the screen, you must add eleven to the y coordinate and one to the x coordinate in order to get your locations to coordinate with the x and y coordinates used in the other BASIC graphics commands (LINE, CIRCLE, GET, PUT, et.al.).

One more major point that you may have noticed, when I DECLARE FUNCTION SetDrMd LIBRARY, I declared the function as a SHORT variable and then I called it as a LONG (SetDrMd&()). The actual declaration does not need to specify it as LONG as it is only looking for the function name in the .bmap file. In fact, by declaring the function as a LONG, it has been seen to cause a visit from the GURU! This does not occur all of the time, but why invite disaster?

The SetDrMd() routine allows you to access the various drawing modes in the library. These modes are JAM1, JAM2, COMPLEMENT and INVERSE. When you are normally printing to the screen, you are in JAM2 mode. I will not go into detail about the different modes here, but they are all available using the SetDrMd() routine. The syntax is:

CALL SetDrMd&(Rp&,Mode%)

where Rp& is, once again, the rastPort address pointer and Mode% is the value of the drawing mode you wish to use (0,1,2,4 or any bitwise combination).

These functions are used to place the text into non-LOCATE oriented positioning to make for a more aesthetic display. Also, I use SetDrMd() when COMPLEMENTING the Requestor selection buttons.

My Requestor routine is being presented as a separate article and includes a complete discussion of the inner workings and variable assignments to allow you to make use of it in your programs.

Running BmapReader

When run, BmapReader will display a Requestor and ask you for a filename which it then searches for in the current or specified directory. When you enter the filename, it isn't necessary to add the .bmap extension as the program will append it for you if it is missing (keeps you from bombing things out by trying to examine the wrong file).

If the file is not found, the requestor returns, informs you of this fact, and asks if you wish to try another name or exit.

If the file is located, the program then displays 20 lines of information based on what was discovered in the file.

The things that are displayed are:

The routine Name
The offset vector jump address (how far from the Base address of the library does the function reside)
The Registers used by the function. (# signs)

I realize that many of you will not find a true use for this program, but it is being presented as a demonstration of library function calls and as a help to programmers that are using the libraries in their programs.

If you have questions or comments, I can be reached through the following BBS's:

People Link - as AmSoft 1
The Window - (617)-868-1430 as myself or Sysop
Wonderland - (617)-665-3796 as myself

Listing 1

```
Initialize:
  FALSE = 0 : TRUE = -1 ' Just because!
  BobRight = 230 : BobBottom = 90 ' The sides of our
requestor
  DEF FNArraySize& = 3 + INT((BobRight +
16)/16)*(BobBottom+1)*2
  DECLARE FUNCTION Move LIBRARY
  DECLARE FUNCTION SetDrMd LIBRARY
  LIBRARY "graphics.library"
  Title$ = "   .bmap Reader                          Tim Jones "
  Title$ = Title$ + CHR$(169) + " 1986 AmSoft Developement"
  WINDOW 1,Title$,(0,0)-(631,186),0
  PALETTE 0,0,0,0
  PALETTE 3,.8,.2,.1
  PALETTE 1,.1,.7,.1
  PALETTE 2,.9,.9,.1
  Rp& = WINDOW(8)
Start:
  COLOR 1,0 : CLOSE 1
  GOSUB NameRequestor
  IF NOT Okay THEN
    CLS : LIBRARY CLOSE
    WINDOW 1,"BmapReader",(0,0)-(617,186),15,-1 : STOP
  END IF
  ON ERROR GOTO FileProb ' this screws things up if you use
other than the
                         ' default workbench screen
  IF UCASE$(RIGHT$(FileName$,5)) <> ".BMAP" THEN
    FileName$ = FileName$ + ".bmap"
  END IF
  OPEN FileName$ FOR INPUT AS 1
  Prompt$ = "" : Prompt2$ = " Output to Printer?"
  GOSUB YNRequestor : IF Okay THEN fPrt = TRUE : GOTO
Printer
  CLS
  LINE(3,13)-(628,170),1,b
  LINE(4,13)-(627,170),1,b
  Length& = LOF(1)
  LOCATE 23,1 : PRINT Length&;"Bytes read.   FILE: ";: COLOR
2,0 : PRINT
FileName$;
  CALL Move&(Rp&,10,10) : COLOR 3,0
  PRINT "Routine Name          Address    d0 d1 d2 d3 d4 d5
d6 d7   a0 a1 a2 a3
a4"
  COLOR 1,0
GetTheFile:
  WHILE NOT EOF(1)
    FOR L = 3 TO 21
    IF EOF(1) THEN
      FOR J = L TO 21
        LOCATE J,2
        PRINT SPACE$(75)
      NEXT J
      GOTO Finished
    END IF
    GOSUB GetRoutName
      COLOR 1,0
      LOCATE L,2
      PRINT " ";Routine$
    GOSUB GetEntryAdd
      LOCATE L,25
      IF LEN(Address$(2)) = 1 THEN
        Address$(2) = "0" + Address$(2)
      END IF
      PRINT Address$(1);Address$(2)
    GOSUB GetRegInfo
      LOCATE L,35
      PRINT
    NEXT L
    COLOR 0,3
    LINE(386,173)-(612,185),3,bf
    LINE(388,174)-(610,184),0,b
    CALL Move&(Rp&,394,182) : PRINT "F1 continues <> F10
aborts";
    WaitKey:
      In$ = INKEY$ : IF In$ = "" THEN WaitKey
      IF In$ = CHR$(138) THEN
        LINE(386,173)-(612,185),0,bf
        GOTO Finished
      END IF
```

```
      IF In$ <> CHR$(129) THEN WaitKey
      COLOR 1,0
      LINE(386,173)-(612,185),0,bf
  WEND
  GOTO Finished

GetRoutName:
  Routine$ = ""
  GOSUB GetChar
  WHILE Char$ <> CHR$(0)
    Routine$ = Routine$ + Char$
    GOSUB GetChar
  WEND
  IF LEN(Routine$) < 30 THEN
    Routine$ = Routine$ + SPACE$(20 - LEN(Routine$))
  END IF
RETURN

GetEntryAdd:
  FOR ii = 1 TO 2
    GOSUB GetChar
    Address$(ii) = HEX$(ASC(Char$))
  NEXT ii
RETURN

GetRegInfo:
  LOCATE L,35 : PRINT SPACE$(42);
  WHILE Char$ <> CHR$(0)
    GOSUB GetChar
    COLOR 2,0
    Register = ASC(Char$)
    GOSUB R1
  WEND
  IF fPrt THEN PRINT #4," "
RETURN

GetChar:
  IF NOT EOF(1) THEN Char$ = INPUT$(1,1)
RETURN

Finished:
  COLOR 3,0
  LOCATE 23,1
  PRINT SPACE$(78);
  LOCATE 23,1
  PRINT " Do you wish to examine another .BMAP file
(Y/N)?";
test:
  In$ = INKEY$ : IF In$ = "" THEN test
  IF UCASE$(In$) <> "Y" THEN
    CLS : CLOSE 1 : LIBRARY CLOSE : WINDOW CLOSE 1
    WINDOW 1,"BmapReader",(0,0)-(617,186),15,-1 : STOP
  END IF
  GOTO Start

R1:
  IF Register < 1 THEN RETURN
  IF Register > 8 THEN R2
  IF fPrt THEN
    PRINT #4,CHR$(141);TAB(32 + (Register * 3));"#";
    RETURN
  END IF
  LOCATE L,(32 + (Register * 3))
  PRINT "#"
RETURN

R2:
  IF fPrt THEN
    PRINT #4,CHR$(141);TAB(34 + (Register * 3));"#";
    RETURN
  END IF
  LOCATE L,(34 + (Register *3))
  PRINT "#"
RETURN

Printer:

  OPEN "LPT1:BIN" FOR OUTPUT AS 4
  PRINT #4,CHR$(14);"Contents of file ";FileName$
  PRINT #4," "
  WHILE NOT EOF(1)
    PRINT #4,"Routine Name          Address     d0 d1 d2 d3 d4
d5 d6 d7   a0 a1 a2
a3 a4"
    PRINT #4,"---
---------          -------    -- -- -- -- -- -- -- --   -- -- -
-
-- --"
    FOR L = 1 TO 54
    IF EOF(1) THEN
      GOTO Finished
    END IF
    GOSUB GetRoutName
      PRINT #4," ";Routine$;
    GOSUB GetEntryAdd
      IF LEN(Address$(2)) = 1 THEN
        Address$(2) = "0" + Address$(2)
      END IF
      PRINT #4,"  ";Address$(1);Address$(2);
    GOSUB GetRegInfo
    NEXT L
    PRINT #4,CHR$(12)
  WEND
  PRINT #4,CHR$(12)
  CLOSE 4 : fPrt = FALSE
  GOTO Start

FileProb:

  flag = ERR
  Prompt$ = ""
  Prompt2$ = "   Error! >>"+ STR$(ERR)
  GOSUB YNRequestor
  IF NOT Okay THEN
    LIBRARY CLOSE
    CLOSE 1
    WINDOW CLOSE 1
    WINDOW 1,"BmapReader",(0,0)-(617,186),31,-1
    END
  END IF
  RESUME Start

NameRequestor:
  Size& = FNArraySize&\2
  DIM ScrSav&(Size&)
  GET(40,40)-(230,90),ScrSav&
DrawRequestorToScreen2:
  LINE(40,40)-(230,90),1,bf    'Main requestor box
  LINE(40,40)-(230,90),0,b     'outline for main requestor
box
  LINE(44,42)-(226,88),0,b     'secondary outline for main
box
  LINE(50,74)-(72,86),3,bf     'OK button box
  LINE(50,74)-(72,86),0,b      'OK outline
  LINE(150,74)-(220,86),3,bf   'CANCEL button box
  LINE(150,74)-(220,86),0,b    'CANCEL outline
  CALL Move&(Rp&,53,83)        'Position for printing OK in
button
  COLOR 0,3 : PRINT "OK"       'print it
  CALL Move&(Rp&,160,83)       'Position for printing CANCEL
button
  COLOR 0,3 : PRINT "CANCEL"   'print it
  LINE(53,50)-(216,62),3,b
  Curs = 55 : LINE(Curs,52)-(Curs+7,60),2,bf  ' Print the
pseudo-cursor
  CALL Move&(Rp&,53,71) : COLOR 0,1 : PRINT "   Enter File
Name"
  C$ = INKEY$ : WHILE C$ <> "" : C$ = INKEY$ : WEND 'Empty
keyboard buffer
  FileName$ = ""

AccessLoop: ' Wait for click in string box or CANCEL

  I = MOUSE(0) : X = MOUSE(1) : Y = MOUSE(2)
  IF I <> 0 THEN
    WHILE I <> 0 : I = MOUSE(0) : X = MOUSE(1) : Y =
MOUSE(2) : WEND
    Y=Y-1 ' This is due to a difference in MOUSE(2) and the
actual Window
          ' location
    IF X > 150 AND X < 220 AND Y > 74 AND Y < 86 THEN '
Check for CANCEL
      CALL SetDrMd&(Rp&,2) : LINE(151,75)-(219,85),0,bf
      CALL SetDrMd&(Rp&,1)
      Okay = FALSE : FOR Delay = 1 TO 1000 : NEXT Delay
      PUT(40,40),ScrSav&,PSET
      ERASE ScrSav& : COLOR 1,0 : RETURN
    END IF
    IF X > 53 AND X < 216 AND Y > 50 AND Y < 62 THEN
      LINE(Curs,52)-(Curs+7,60),0,bf
```

```
      FOR Delay = 1 TO 50 : NEXT Delay
      LINE(Curs,52)-(Curs+7,60),2,bf
      WHILE INKEY$ <> "" : WEND
      GOTO Loop
    END IF
  END IF
  GOTO AccessLoop

Loop: ' We do this until CANCEL, OK or Carriage Return

  C$ = INKEY$ : I = MOUSE(0) : X = MOUSE(1) : Y = MOUSE(2)
  IF I <> 0 THEN
    WHILE I <> 0 : I = MOUSE(0) : X = MOUSE(1) : Y =
MOUSE(2) : WEND
    Y=Y-1 ' This is due to a difference in MOUSE(2) and the
actual Window
          ' location
    IF X > 150 AND X < 220 AND Y > 74 AND Y < 86 THEN '
Check for CANCEL
      CALL SetDrMd&(Rp&,2) : LINE(151,75)-(219,85),0,bf
      CALL SetDrMd&(Rp&,1)
      Okay = FALSE : FOR Delay = 1 TO 1000 : NEXT Delay
      PUT(40,40),ScrSav&,PSET
      ERASE ScrSav& : COLOR 1,0 : RETURN
    END IF
    IF X > 50 AND X < 72 AND Y > 74 AND Y < 86 AND
LEN(FileName$) > 0 THEN
      ' Check for OK and length of file
      CALL SetDrMd&(Rp&,2) : LINE(51,75)-(71,85),0,bf
      CALL SetDrMd&(Rp&,1)
      FOR Delay = 1 TO 1000 : NEXT Delay : Okay = TRUE :
fExist = TRUE
      PUT(40,40),ScrSav&,PSET
      ERASE ScrSav& : COLOR 1,0 : RETURN
    END IF
  END IF
  IF C$ = "" THEN GOTO Loop
  IF LEN(FileName$) = 0 THEN IF C$ < "A" AND ASC(C$) <> 13
GOTO Loop
    ' Don't allow non-Alpha characters as first character
  IF ASC(C$) = 13 THEN
    Okay = TRUE : fExist = TRUE
    PUT(40,40),ScrSav&,PSET
    ERASE ScrSav& : COLOR 1,0 : RETURN
  END IF
  IF ASC(C$) = 8 THEN
    ' Capture the BackSpace and fix display and filename
    FileName$ = LEFT$(FileName$,LEN(FileName$)-1)
    LINE(Curs,52)-(Curs+7,60),1,bf
    Curs = Curs-8 : LINE(Curs,52)-(Curs+7,60),2,bf
    GOTO Loop
  END IF
  IF LEN(FileName$) = 19 THEN GOTO Loop
  IF ASC(C$) = 8 THEN Loop
  IF C$ < " " OR (C$ > "Z" AND C$ < "a") OR C$ > "z" GOTO
Loop
  FileName$ = FileName$ + C$
  LINE(Curs,52)-(Curs+7,60),1,bf
  COLOR 0,1 : CALL Move&(Rp&,0,59) : PRINT PTAB(Curs);C$;
  Curs = Curs + 8 : LINE(Curs,52)-(Curs+7,60),2,bf
  GOTO Loop

YNRequestor:

  Size& = FNArraySize&\2 'to reserve memory for the GET
statement
  DIM ScrSav&(Size&)
  GET(40,40)-(230,90),ScrSav&
  LINE(40,40)-(230,90),2,bf    'Main requestor box
  LINE(40,40)-(230,90),0,b
  LINE(44,42)-(226,88),0,b
  LINE(50,74)-(72,86),3,bf     'OK button box
  LINE(50,74)-(72,86),0,b      'OK outline
  LINE(150,74)-(220,86),3,bf   'CANCEL button box
  LINE(150,74)-(220,86),0,b    'CANCEL outline
  CALL Move&(Rp&,53,83)        'Position for OK in button
  COLOR 0,3 : PRINT "OK"       'print it
  CALL Move&(Rp&,160,83)       'Position for CANCEL button
  COLOR 0,3 : PRINT "CANCEL"   'print it
  CALL Move&(Rp&,54,52)        'Position for first text
  COLOR 0,2 : PRINT Prompt$    'print it
  CALL Move&(Rp&,54,62)        'Position for second text
  COLOR 3,2 : PRINT Prompt2$   'print it

GetButton2:

  ' This waits for a mouse click (left mouse button)

  I = MOUSE(0) : X = MOUSE(1) : Y = MOUSE(2)
  IF I <> 0 THEN
    WHILE I<>0 : I = MOUSE(0) : X = MOUSE(1) : Y = MOUSE(2)
: Y = Y-1
    WEND
Cancel: ' Check to see if the CANCEL button is selected
    IF X > 150 AND X < 220 AND Y > 74 AND Y < 86 THEN
      CALL SetDrMd&(Rp&,2)
      LINE(151,75)-(219,85),0,bf
      CALL SetDrMd&(Rp&,1)
      FOR Delay = 1 TO 1000 : NEXT Delay
      PUT(40,40),ScrSav&,PSET
      ERASE ScrSav&            ' Erase the Array
      Okay = FALSE             '
      COLOR 1,0
      RETURN
    END IF
Ok:  ' Check to see if the OK button is selected
    IF X > 50 AND X < 72 AND Y > 74 AND Y < 86 THEN
      CALL SetDrMd&(Rp&,2)   ' COMPLIMENT the OK button
      LINE(51,75)-(71,85),0,bf
      CALL SetDrMd&(Rp&,1)
      FOR Delay = 1 TO 1000 : NEXT Delay
      PUT(40,40),ScrSav&,PSET
      Okay = TRUE                    ' Erase the Array
      ERASE ScrSav&
      COLOR 1,0
      RETURN
    END IF
  END IF
  GOTO GetButton2 ' Until a button is selected
```

•AC•

Last, but not least.....

An Amiga Training Video and an RF Modulator, a quick look at two products received

The Amiga on Video

Dan Sorensen and Doug Allen of Clackamas Computers in Clackamas, Oregon have teamed with Aurion Video Productions to produce a ninety minute video tape titled Introduction to the Amiga.

For most users who have already purchased their machines, set them up, and struggled through the efforts of learning Workbench and CLI, the purchase is probably not necessary. However, if you are teaching your personnel, a group of students, your customers, or yourself from a book on CLI for the tenth time, it is an ideal purchase.

The Video is presented in three parts:
Setting-up and Caring for your Amiga,
hosted by Dan Sorensen
Using the Workbench, hosted by Dr. Doug Allen
Introduction to CLI hosted by Dr. Doug Allen

Caring for the Amiga

The set up and caring portion is definitly for the first time computer user. Dan Sorensen takes the user step by step through the setting up of the Amiga. There are also portions on cleaning disk drives, surge protectors and other information that the first time user will find informative.

Workbench

The Using the Workbench section is a great deal more thorough. However, most users who have spent some time with their units will find this information repetitive.

Dr. Doug Allen spends a great deal of time and care in leading the user through the different intricacies of the Amiga. When a user first enters this area, this section will be greatly appreciated. Dr. Allen leads users through the seperate Icons, preferences, and disk copy.

CLI

In the third and final section, Introduction to CLI, Dr. Allen prepares the user through the difficult area of the Amiga operating system. From our experience at PiM, there are a great deal of users who have never attempted to learn about this almost undocumented area.

Dr. Allen does what Commodore did not. He teaches the overall ins and outs of using the most powerful part of the Amiga operating system by carefully leading the user through the more difficult operations. By carfuly demonstrating Directories, Ram Disk, and some function commands, Dr. Allen makes CLI a bit more understandable then a text alone.

Last notes

Again, this tape is not for everyone. However, I recommend it for Amiga Dealers who want to provide good support for the Amiga without spending a great deal of time teaching the everyday items of the Amiga to all of their customers.

The tape can also be useful to users who either become overwhelmed by a text presentation of CLI or would rather have the one on one demonstration ability of a video tape. Remember, if you don't understand it the first time, you can repeat the section of the tape.

This Video will not answer all questions on CLI but it will get a new user *online*.

Introduction to the Amiga
$29.95
Clackamas Computers
16234 S.E. 82nd Dr.
Clackamas, Oregon 97015
(503) 650- 0379

An RF Modulator for the Amiga

MJ Products has announced their MJ-Modulator for the Commodore Amiga as "an alternative to the high dollars for an RF-modulator interface".

It uses your television for larger displays (better for large classroom demonstrations)

$22.50
MJ Products
23181 Broadway Avenue
Oakwood Village, OH 44146
1-219-439-3827

•AC•

Index to Advertisers

LOGISTIX

IS

Spreadsheet+Timesheet+Database+Graphics+Project Management=

Successful Management!

Computer software has always been the limiting factor in your business operation. Now Progressive Peripherals & Software, Inc. has broken through this barrier to bring you a completely new concept in integrated business software. *Logistix*, the software that outweighs all other business software products!

Logistix is the planning tool designed to provide professional business people with a solution to the limitations of current spreadsheets and project management software. With time and project management built in to the same work area, you can now solve the simultaneous problems of time and money that effect your business for today and tomorrow.

Logistix is the first and only business software program to actually integrate both time and financial functions. Face it: Time is Money. With *Logistix*, you can schedule personnel, manage shipments, plan work schedules and production flows. This allows you to monitor, project, and protect your business interests better than ever before.

Logistix offers you the four dynamic features needed to widen the horizons of success for your business. *Logistix* combines together, in one worksheet area, a large and sophisticated spreadsheet, presentation quality color graphics, database functions and, of course, powerful and flexible time and project management. Best of all, it is all designed with the business person in mind.

Logistix has built-in sideways print functions, supports over 20 international currency symbols, and offers complete support documentation, including a examples diskette. *Logistix* reads 1-2-3 files and many other file formats, so no time is lost reformatting existing data. You'll be finding business solutions faster than ever before.

Logistix is the most modern and intelligent planning tool ever designed for business people, by business people. It's loaded with features you expect to find in programs costing hundreds of dollars more. When it comes to business software, *Logistix* helps you see the future possibilities of your business.

For more information about *Logistix* call or write us today.

AMIGA

IBM

HEWLETT PACKARD

WHEN QUALITY COUNTS!

PROGRESSIVE PERIPHERALS & SOFTWARE

464 KALAMATH STREET
DENVER, COLORADO 80204
303-825-4144
TELEX: 888837

UNLEASH THE AWESOME POWER OF THE AMIGA!

PAL SYSTEMs. The PAL is a turnkey expansion chassis that provides the most powerful and cost effective hardware growth path for your Amiga

Standard Configuration. 1 Megabyte RAM. This meg of RAM is located in the Amiga's reserve memory and will autoconfigure. Note that this memory is in addition to the 8 meg of RAM accessible through the expansion memory and does not occupy any of the 5 DMA expansion slots. Battery-backed clock/calendar. Built into chassis motherboard - does not occupy any of the 5 DMA expansion slots. 5 DMA expansion slots. Conforms to ZORRO Expansion architecture (100 pin connectors). Plug in cards mounted via card cage - cards accessible from PAL's rear panel. No need to open the Pal System to access or insert cards. Internally supports up to 3 half height storage/ retrieval devices. Chassis will autoconfigure under kickstart/workbench 1.2. 200 watt power supply. Chassis rests on top of the Amiga. Chassis length and width are identical to that of the Amiga. Height is approximately 5 inches. Attractive aluminum casing styled in the Amiga profile. Whisper fan for cooling.

Chassis with 20 MEG DMA hard disk. High speed DMA hard disk controller is capable of transmitting data at 10 MegaBits per second. Supports two ST506/412 storage/retrieval devices. Optional SCSI controller available within 4 weeks of release. The 20 Meg hard disk has an average access time of 65 Milliseconds and track-to-track access of 18 Milliseconds.

Chassis with 40 MEG DMA hard disk. High speed DMA hard disk controller - transmits data at 10 MegaBits per second. The 40 Meg hard disk has an average access time of 35 Milliseconds and track-to-track access of 6 Milliseconds.

Optional expansion cards

2 Megabyte RAM card. High speed memory board incorporating SIMM technology. Upgradeable to 8 megabytes on this board using 1 megabyte chips.

Prototyping cards

512K RAM daughterboard with parity. Plugs into the PAL motherboard-doesn't take up an expansion slot. 512K in Amiga reserve memory. Fully autoconfigures. Provides parity for this 512K and one meg ram on the PAL's boards.

PAL JR. combines a 20 MEG hard disk and a DMA SCSI controller with one Megabyte of fast ram. PAL JR.'s low profile casing, styled in Amiga colors, conveniently connects to the Amiga's buss and extends the SCSI port to provide further expansion. The small footprint saves valuable real estate while providing you with unparalleled power to accomplish serious work. Naturally, PAL JR. autoconfigures under release 1.2 of the operating system. Simply connect PAL JR. and unleash the awesome power of your Amiga! Best of all, the suggested retail price is only $1,495.

INFOMINDER is an intelligent information resource that provides the user with instantaneous access to reference information stored within the Amiga personal computer. Think of INFOMINDER as an electronic library. Fully supports multi-tasking, Fast access by menu or outline. Expand and Shrink topics with a simple mouse click. Text capabilities include: Justification, Word Wrap, Multiple character font/styles. Information content completely user defineable. Supports combination of TEXT and IFF GRAPHICS. Programmatic interface for context sensitive help. Narration and printing of information.

Use INFOMINDER to hierarchially organize and display pictoral files, i.e. Real Estate Listings, Personnel Files, Digitized X-RAYS, product descriptions, collections, etc.

INFOMINDER is the ideal access mechanism for CD ROM and Interactive Laser Disk technology. Imagine instantaneous access to entire libraries, i.e. encylopedias, law libraries, etc.

INFOMINDER is revolutionizing the way we store and access textual and graphical information. Stop searching and START using the information around you. Get INFOMINDER today at the special introductory price of only $89.95.

TIC. The TIC provides your Amiga with a tiny battery backed clock/calendar. Conveniently plugs into the second joystick port. The TIC's 3 year battery will maintain time even if temporarily removed from the Amiga. Change the Amiga's internal time simply by moving the displayed clock's hands with the mouse. Set your Amiga's time once and for all. It's about time for the TIC. Suggested retail only $59.95.

Arboretum Plaza II 9442 Capitol of Texas Highway BYTE by BYTE CORPORATION Suite 150 Austin, TX 78759 (512) 343-4357